Enforcing Secure and Privacy-Preserving Information Brokering in Distributed Information Sharing

Fengjun Li, Bo Luo, Peng Liu Dongwon Lee and Chao-Hsien Chu

***Abstract*—Today's organizations raise an increasing need for information sharing via on-demand access. Information Brokering Systems (IBSs) have been proposed to connect large-scale loosely-federated data sources via a brokering overlay, in which the brokers make routing decisions to direct client queries to the requested data servers. Many existing IBSs assume that brokers are trusted and thus only adopt server-side access control for data confidentiality. However, privacy of data location and data consumer can still be inferred from metadata (such as query and access control rules) exchanged within the IBS, but little attention has been put on its protection. In this article, we propose a novel approach to preserve privacy of multiple stakeholders involved in the information brokering process. We are among the first to formally define two privacy attacks, namely *attribute-correlation attack* and *inference attack*, and propose two countermeasure schemes *automaton segmentation* and *query segment encryption* to securely share the routing decision making responsibility among a selected set brokering servers. With comprehensive security analysis and experimental results, we show that our approach seamlessly integrates security enforcement with query routing to provide system-wide security with insignificant overhead.**

***Index Terms*—Access control, information sharing, privacy**

I. Introduction

Along with the explosion of information collected by organizations in many realms ranging from business to government agencies, there is an increasing need for inter-organizational information sharing to facilitate extensive collaboration. While many efforts have been devoted to reconcile data heterogeneity and provide interoperability, the problem of balancing peer autonomy and system coalition is still challenging. Most of the existing systems work on two extremes of the spectrum, adopting either the query-answering model to establish pairwise client-server connections for on-demand information access, where peers are fully autonomous but there lacks system-wide coordination, or the distributed database model, where all peers with little autonomy are managed by a unified DBMS.

Unfortunately, neither model is suitable for many newly emerged applications, such as healthcare or law enforcement information sharing, in which organizations share information in a conservative and controlled manner due to business considerations or legal reasons. Take healthcare information systems as example. Regional Health Information Organization (RHIO) [1] aims to facilitate access to and retrieval of clinical data across collaborative healthcare providers that include a number of regional hospitals, outpatient clinics, payers, etc. As a data provider, a participating organization would not assume free or complete sharing with others, since its data is legally private or commercially proprietary, or both. Instead, it requires to retain full control over the *data* and the *access to the data*. Meanwhile, as a consumer, a healthcare provider requesting data from other providers expects to preserve her privacy (e.g., identity or interests) in the querying process.

In such a scenario, sharing a complete copy of the data with others or "pouring" data into a centralized repository becomes impractical. To address the need for autonomy, federated database technology has been proposed [2], [3] to manage locally stored data with a federated DBMS and provide unified data access. However, the centralized DBMS still introduces data heterogeneity, privacy, and trust issues. While being considered a solution between "sharing nothing" and "sharing everything", peer-to-peer information sharing framework essentially need to establish pairwise client-server relationships between each pair of peers, which is not scalable in large scale collaborative sharing.

In the context of sensitive data and autonomous data providers, a more practical and adaptable solution is to construct a data-centric overlay (e.g., [4], [5]) consisting of data sources and a set of brokers that make routing decisions based on the content of the queries [6], [7], [8], [9]. Such infrastructure builds up semantic-aware index mechanisms to route the queries based on their content, which allows users to submit queries without knowing data or server location. In our previous study [9], [10], such a distributed system providing data access through a set of brokers is referred to as *Information Brokering System* (IBS). As shown in Figure 1, applications atop IBS always involve some sort of consortium (e.g., RHIO) among a set of organizations. Databases of different organizations are connected through a set of brokers, and metadata (e.g. data summary, server locations) are "pushed" to the *local brokers*, which further "advertise" (some of) the metadata to other brokers. Queries are sent to the local broker and routed according to the metadata until reaching the right data server(s). In this way, a large number of information sources in different organizations are loosely federated to provide an unified, transparent, and on-demand data access.

While the IBS approach provides scalability and server autonomy, privacy concerns arise, as brokers are no longer assumed fully trustable – the broker functionality may be outsourced to third-party providers and thus vulnerable to be

F. Li and B. Luo are with the Department of EECS, The University of Kansas, Lawrence, KS, 66045 USA. E-mail: {fli,bluo}@ku.edu.

P. Liu, D. Lee and C. Chu are with the College of IST, The Pennsylvania State University. Email: {pliu,dlee,chu}@ist.psu.edu.

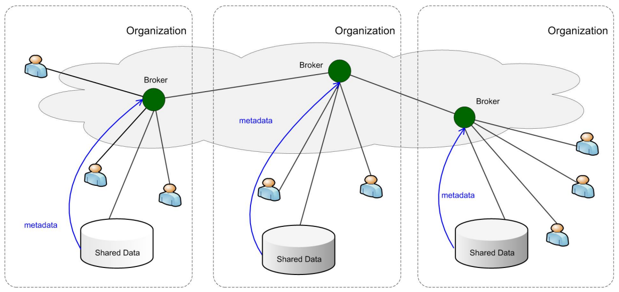


Fig. 1. An overview of the IBS infrastructure.

abused by insiders or compromised by outsiders.

In this article, we present a general solution to the privacy-preserving information sharing problem. First, to address the need for privacy protection, we propose a novel IBS, namely *Privacy Preserving Information Brokering* (PPIB). It is an overlay infrastructure consisting of two types of brokering components, *brokers* and *coordinators*. The brokers, acting as mix anonymizer [11], are mainly responsible for user authentication and query forwarding. The coordinators, concatenated in a tree structure, enforce access control and query routing based on the embedded non-deterministic finite automata – the *query brokering automata*. To prevent curious or corrupted coordinators from inferring private information, we design two novel schemes to segment the query brokering automata and encrypt corresponding query segments so that routing decision making is decoupled into multiple correlated tasks for a set of collaborative coordinators. while providing integrated in-network access control and content-based query routing, the proposed IBS also ensures that a curious or corrupted coordinator is not capable to collect enough information to infer privacy, such as “which data is being queried”, “where certain data is located”, or “what are the access control policies”, etc. Experimental results show that PPIB provides comprehensive privacy protection for on-demand information brokering, with insignificant overhead and very good scalability.

The rest of the paper is organized as follows: we discuss the privacy requirements and threats in the information brokering scenario in Section II, and introduce the related work and preliminaries in Section III. In Section IV, we present two core schemes and the PPIB approach. We discuss the construction and maintenance in Section V, analyze system privacy and security in Section VI, evaluate the performance in Section VII, and conclude our work in Section VIII.

II. The Problem

A. Vulnerabilities and the Threat model

In a typical information brokering scenario, there are **three** types of stakeholders, namely *data owners*, *data providers*, and *data requestors*. Each stakeholder has its own privacy: (1) the privacy of a data owner (e.g., a patient in RHIO) is the identifiable data and sensitive or personal information carried by this data (e.g., medical records). Data owners usually sign strict privacy agreements with data providers to prevent unauthorized use or disclosure. (2) Data providers store the collected data locally and create two types of metadata, namely *routing metadata* and *access control metadata*, for data brokering. Both types of metadata are considered privacy of a data provider. (3) Data requestors may reveal identifiable or private information (e.g., information specifying her interests) in the querying content. For example, a query about AIDS treatment reveals the (possible) disease of the requestor.

We adopt the *semi-honest* [12] assumption for the brokers, and assume two types of adversaries, *external attackers* and *curious or corrupted brokering components*. External attackers passively eavesdrop communication channels. Curious or corrupted brokering components, while following the protocols properly to fulfill brokering functions, try their best to infer sensitive or private information from the querying process.

Privacy concerns arise when identifiable information is disseminated with no or poor disclosure control. For example, when data provider pushes routing and access control metadata to the local broker [6], [9], a curious or corrupted broker learns *query content* and *query location* by intercepting a local query, *routing metadata* and *access control metadata* of local data servers and from other brokers, and *data location* from routing metadata it holds. Existing security mechanisms focusing on confidentiality and integrity cannot preserve privacy effectively. For instance, while data is protected over encrypted communication, external attackers still learn *query location* and *data location* from eavesdropping. Combining types of unintentionally disclosed information, the attacker could further infer the privacy of different stakeholders through *attribute-correlation attacks* and *inference attacks*.

Attribute-correlation attack. Predicates of an XML query describe conditions that often carry sensitive and private data (e.g., name, SSN, creditcard number, etc.) If an attacker intercepts a query with multiple predicates or composite predicate expressions, the attacker can “correlate” the attributes in the predicates to infer sensitive information about data owner. This is known as the *attribute correlation attack*.

Example 1. A tourist Anne is sent to ER at California Hospital. Doctor Bob queries for her medical records through a medicare IBS. Since Anne has the symptom of leukemia, the query contains two predicates: `[pName="Anne"]`, and `[symptom="leukemia"]`. Any malicious broker that has helped routing the query could guess “Anne has a blood cancer” by correlating the two predicates in the query. □

Unfortunately, query content including sensitive predicates cannot be simply encrypted since such information is necessary for content-based query routing. Therefore, we are facing a paradox of the requirement for content-based brokering and the risk of attribute-correlation attacks.

Inference attack. More severe privacy leak occurs when an attacker obtains more than one type of sensitive information and learns explicit or implicit knowledge about the stakeholders through association. By “implicit”, we mean the attacker infers the fact by “guessing”. For example, an attacker can guess the identity of a requestor from her query location (e.g., IP address). Meanwhile, the identity of the data owner could be explicitly learned from query content (e.g., name or SSN). Attackers can also obtain publicly-available information to

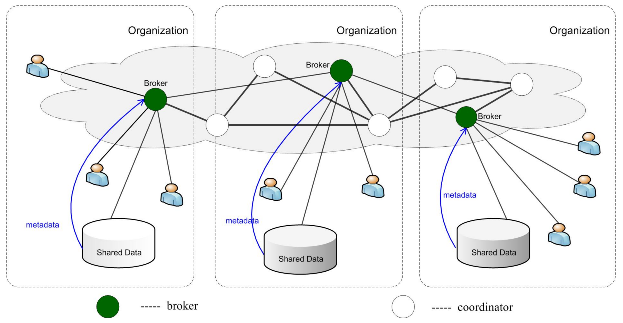


Fig. 2. The architecture of PPIB.

help his inference. For example, if an attacker identifies that a data server is located at a cancer research center, he can tag the queries as "cancer-related".

In summary, we have three reasonable inferences from three distinct combinations of private information: (1) from *query location & data location*, the attacker infers about *who* (i.e., a specific requestor) is interested in *what* (i.e., a specific type of data). (2) From *query location & query content*, the attacker infers about *where who* is, or *who* is interested in *what* (if predicates describe symptom or medicine, etc), or *something* about the data owner (if predicate identifies name or address of a personnel), etc. (3) From *query content & data location*, the attacker infers *which* data server has *which* data. Hence, the attacker could continuously create artificial queries or monitor user queries to learn the data distribution of the system, which could be used to conduct further attacks.

B. Solution Overview

To address the privacy vulnerabilities in current information brokering infrastructure, we propose a new model, namely *Privacy Preserving Information Brokering* (PPIB). PPIB has three types of brokering components: *brokers*, *coordinators*, and a *central authority* (CA). The key to preserving privacy is to divide and allocate the functionality to multiple brokering components in a way that no single component can make a meaningful inference from the information disclosed to it.

Figure 2 shows the architecture of PPIB. Data servers and requestors from different organizations connect to the system through local brokers (i.e., the green nodes in Fig. 2). Brokers are interconnected through coordinators (i.e., the white nodes). A local broker functions as the "entrance" to the system. It authenticates the requestor and hides his identity from other PPIB components. It would also permute query sequence to defend against local traffic analysis.

Coordinators are responsible for content-based query routing and access control enforcement. With privacy-preserving considerations, we cannot let a coordinator hold any rule in the complete form. Instead, we propose a novel *automaton segmentation scheme* to divide (metadata) rules into segments and assign each segment to a coordinator. Coordinators operate collaboratively to enforce secure query routing. A *query segment encryption scheme* is further proposed to prevent coordinators from seeing sensitive predicates. The scheme divides a query into segments, and encrypts each segment in a way that to each coordinator enroute only the segments that are needed for secure routing are revealed. Last but not least, we assume a separate *central authority* handles key management and metadata maintenance.

III. Background

A. Related Work

Research areas such as information integration, peer-to-peer file sharing systems and publish-subscribe systems provide partial solutions to the problem of large-scale data sharing. Information integration approaches focus on providing an integrated view over a large number of heterogeneous data sources by exploiting the semantic relationship between schemas of different sources [13], [14], [15]. The PPIB study assumes that a global schema exists within the consortium, therefore, information integration is out of our scope.

Peer-to-peer systems are designed to share files and data sets (e.g., in collaborative science applications). Distributed hash table technology [16], [17] is adopted to locate replicas based on keyword queries. However, although such technology has recently been extended to support range queries [18], the coarse granularity (e.g. files and documents) cannot meet the expressiveness needs of applications focused in this work. Furthermore, P2P systems often returns an incomplete set of answers while we need to locate all relevant data in the IBS.

Addressing a conceptually dual problem, XML publish-subscribe systems (e.g., [19], [20]) are probably the closely related technology to the proposed research problem: while PPIB aims to locate relevant data sources for a given query and route the query to these data sources, the pub/sub systems locate relevant consumers of a given document and route the document to these consumers. However, due to this duality, we have different concerns. The pub/sub systems focus more on efficiently delivering the same piece of information to a large number of consumers, while we are trying to route a large volume but small-sized queries to fewer sites. Accordingly, the multicast solution in pub/sub systems does not scale in our environment and we need to develop new mechanisms.

One idea is to build an XML overlay architecture that supports expressive query processing and security checking atop normal IP network. In particular, specialized data structures are maintained on overlay nodes to route XML queries. In [5], a robust mesh has been built to effectively route XML packets by making use of self-describing XML tags and the overlay networks. Kouds et al. also proposed a decentralized architecture for ad hoc XPath query routing across a collection of XML databases [6]. To share data among a large number of autonomous nodes, [21] studied content-based routing for path queries in peer-to-peer systems. Different from these approaches, PPIB seamlessly integrates query routing with security and privacy protection.

Privacy concerns arise in inter-organizational information brokering since one can no longer assume brokers controlled by other organizations are fully trustable. As the major source that may cause privacy leak is the metadata (i.e., indexing and access control), secure index based search schemes [22], [23] may be adopted to outsource metadata in encrypted form to untrusted brokers. Brokers are assumed to enforce security check and make routing decision without knowing the content of both query and metadata rules. Various protocols have been proposed for searchable encryption [22], [24], [23], however, to the best of our knowledge, all the schemes presented so far only support keyword search based on exact matching. While there are approaches proposed for multi-dimensional keyword search [25] and range queries [26], supporting queries with complex predicates (e.g., regular expressions) or structures (e.g., XPath queries) is still a difficult open problem. In terms of privacy-preserving brokering, another related technique is secure computation [27] that allows one party to evaluate various functions on encrypted data without being able to decrypt. Originally designed for privacy information retrieval (PIR) in database systems [28], such schemes have the same limitation that only keyword-based search is supported.

Research on anonymous communication provides a way to protect information from unauthorized parties. Many protocols have been proposed to enable the sender node dynamically select a set of nodes to relay its requests [29], [30]. These approaches can be incorporated into PPIB to protect location of data requestors and data servers from irrelevant or malicious parties. However, aiming at enforcing access control during query routing, PPIB addresses more privacy concerns other than anonymity, and thus faces more challenges.

Finally, research on distributed access control is also related to our work ([31] gives a good overview on access control in collaborative systems). In summary, earlier approaches implement access control mechanisms at the nodes of XML trees and filter out data nodes that users do not have authorization to access [32], [33]. These approaches rely much on the XML engines. View-based access control approaches create and maintain a separate view (e.g., a specific portion of XML documents) for each user [34], [35], which causes high maintenance and storage costs. In this work, we adopt an NFA-based query rewriting access control scheme proposed recently in [36], [9], which has a better performance than previous view-based approaches [33].

B. Preliminaries

1) XML Data Model and Access Control: The eXtensible Markup Language (XML) has emerged as the *de facto* standard for information sharing due to its rich semantics and extensive expressiveness. We assume that all the data sources in PPIB exchange information in XML format, i.e., taking XPath [37] queries and returning XML data. Note that the more powerful XML query language, XQuery, still uses XPath to access XML nodes. In XPath, predicates are used to eliminate unwanted nodes, where test conditions are contained within square brackets "`[ ]`". In our study, we mainly focus on *value-based* predicates.

To specify the authorization at the node level, fine-grained access control models are desired. We adopt the 5-tuple access control policy that is widely used in the literature [38], [34], [36]. The policy consists of a set of access control rules (ACR) = {subject, object, action, sign, type}, where (1) *subject* is the role to whom the authorization is granted; (2) *object* is a set

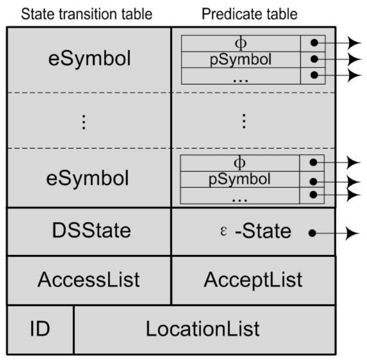


Fig. 3. Data structure of an NFA state.

of XML nodes specified by an XPath expression; (3) *action* is operations as "read", "write", or "update"; (4) *sign* $\in \{+, -\}$ refers to access "granted" or "denied", respectively; and (5) *type* $\in \{LC, RC\}$ denotes "local check" (i.e., applying authorization only to the attributes or textual data of the context nodes) or "recursive check" (i.e., applying authorization to all the descendants of the context node). A set of example rules are shown below:

Example 2. Example ACRs:
```
R₁:{role₁,/site//person/name, read,+,RC}
R₂:{role₁,/site/regions/asia/item,read,+,RC}
R₃:{role₂,/site/regions/asia/item,read,+,RC}
R₄:{role₂,/site/regions/*/item/name,read,+,RC}
R₅:{role₂,/site/regions/*/item[location="USA"]
/description,read,+,RC}
```
□

Existing access control enforcement approaches can be classified as *engine-based* [39], [32], [35], [40], [41], *view-based* [42], [43], [38], [44], [45], *preprocessing* [33], [34], [46], [47], [48], and *post-processing* [49] approaches. In particular, we adopt the *Non-deterministic Finite Automaton* (NFA) based approach as presented in [36], which allows access control to be enforced outside data servers, and independent from the data. The NFA-based approach constructs NFA elements for four building blocks of common XPath axes (`/x`, `//x`, `/*`, and `//*`) so that XPath expressions, as combinations of these building blocks, can be converted to an NFA, which is used to match and rewrite incoming XPath queries. Please refer to [36] for more details on the QFilter approach.

2) Content-based Query Brokering: Indexing schemes have been proposed for content-based XML retrieval [50], [51], [52], [53]. The index describes the address of the data server that stores a particular data item requested by an user query. Therefore, a content-based index rule should contain the *content description* and the *address*. In [9], we presented a content-based indexing model with index rules in the form of I = {object, location}, where (1) *object* is an XPath expression that selects a set of nodes; and (2) *location* is a list of IP addresses of data servers that hold the content.

Example 3. Example index rules:
```
I₁:{/site/people/person/name, 130.203.189.2}
I₂:{/site/regions//item[@id>"100"],
135.176.4.56}
I₃:{/site/regions/samerica/item[@id>"200"],
195.228.155.9}
```

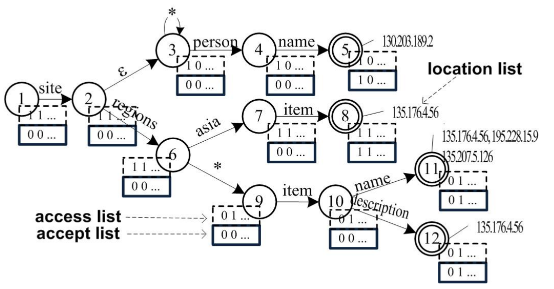


Fig. 4. The state transition graph of the QBroker that integrates index rules with ACRs.

```
I4:{/site//namerica/item/name, 135.207.5.126}
I5:{/site/regions/namerica/item/location,
74.128.5.91}
```
□

When an user queries the system, the XPath query is matched with the *object* field of the index rules, and the matched query will be sent to the data server specified by the *location* field of the rule(s). While other techniques (e.g., bloom filter [7], [6]) can be used to implement content-based indexing, we adopt the model in [9] in our study since it can be directly integrated with the NFA-based access control enforcement scheme. We call the integrated NFA that captures access control rules and index rules *content-based query broker* (QBroker). QBroker is constructed in a similar way as QFilter [36]. Fig. 3 shows the data structure of each NFA state in QBroker, where the state transition table stores the child nodes specified by the XPath expression as the child states in eSymbol. The binary flag DSState indicates that the state is a "double-slash" state. "double-slash" state, whose child state is an ϵ-transition state that directly transits to the next state without consuming any input symbol, will recursively accept input symbols. Fig. 4 shows a QBroker constructed from Example 2 and 3. Unlike QFilter that captures ACRs for only one role, QBroker adds two binary arrays to each state to capture rules for multiple roles: AccessList determines the roles that are allowed to access this state and AcceptList indicates for which role(s) the state is an accept state. For instance, in Fig. 4, the accept list of state 5 is [1 0], indicating the state is an accept state for $\texttt{role}_1$ but not for $\texttt{role}_2$, and the access list of state 6 is [1 1], indicating this state is accessible by both roles. A LocationList is attached to each accept state. In the brokering process, QBroker first checks if a query is allowed to access the requested nodes according to the role type and then makes routing decision. If a query can access only a subset of the requested data, it will be rewritten into a "safe" query before forwarding.

IV. Privacy-preserving query brokering scheme

The QBroker [9] approach has severe privacy vulnerability as we discussed in Section II. If the QBroker is compromised or cannot be fully trusted (e.g., under the honest-but-curious assumption as in our study), the privacy of both requestor and data owner is under risk. To tackle the problem, we present the PPIB infrastructure with two core schemes. In this section, we first explain the details of *automata segmentation* and *query segment encryption* schemes, and then describe the 4-phase query brokering process in PPIB.

A. Automaton Segmentation

In the context of distributed information brokering, multiple organizations join a consortium and agree to share the data within the consortium. While different organizations may have different schemas, we assume a global schema exists by aligning and merging the local schemas. Thus, the access control rules and index rules for all the organizations can be crafted following the same shared schema and captured by a global automaton. The key idea of automaton segmentation scheme is to *logically* divide the global automaton into multiple independent yet connected segments, and *physically* distribute the segments onto different brokering components, known as coordinators.

1) Segmentation: The atomic unit in the segmentation is an NFA state of the original automaton. Each segment is allowed to hold one or several NFA states. We further define the *granularity level* to denote the greatest distance between any two NFA states contained in one segment. Given a granularity level k, for each segmentation, the next i ($\in [1, k]$) states will be divided into one segment with a probability $1/k$. Obviously, with a larger granularity level, each segment will contain more NFA states, resulting in less segments and smaller end-to-end overhead in distributed query processing. However, a coarse partition is more likely to increase the privacy risk. The tradeoff between the processing complexity and the degree of privacy should be considered in deciding the granularity level. As privacy protection is of the primary concern of this work, we suggest a granularity level ≤ 2. To reserve the logical connection between the segments after segmentation, we define the following *heuristic segmentation rules*: (1) NFA states in the same segment should be connected via parent-child links; (2) sibling NFA states should not be put in the same segment without their parent state; and (3) the "accept state" of the original global automaton should be put in separate segments. To ensure the segments are logically connected, we also make the last states of each segment as "dummy" accept states, with links pointing to the segments holding the child states of the original global automaton.

Algorithm 1 The automaton segmentation algorithm: $deploySegment()$

```
Input: Automaton State $S$
Output: Segment Address: $addr$
 1: for each symbol $k$ in $S.StateTransTable$ do
 2:     $addr$=$deploySegment(S.StateTransTable(k).nextState)$
 3:     $DS$=$createDummyAcceptState()$
 4:     $DS.nextState \leftarrow addr$
 5:     $S.StateTransTable(k).nextState \leftarrow DS$
 6: end for
 7: $Seg = createSegment()$
 8: $Seg.addSegment(S)$
 9: $Coordinator = getCoordinator()$
10: $Coordinator.assignSegment(Seg)$
11: return $Coordinator.address$
```

2) Deployment: We employ physical brokering servers, called *coordinators*, to store the logical segments. To reduce the number of needed coordinators, several segments can be deployed on the same coordinator using different port numbers. Therefore, the tuple $<$ coordinator, port $>$ uniquely identifies a segment. For the ease of presentation, we assume each coordinator only holds one segment in the rest of the article. After the deployment, the coordinators can be linked together according to the relative position of the segments they store, and thus form a tree structure. The coordinator holding the root state of the global automaton is the root of the coordinator tree and the coordinators holding the accept states are the leaf nodes. Queries are processed along the paths of the coordinator tree in a similar way as they are processed by the global automaton: starting from the root coordinator, the first XPath step (token) of the query is compared with the tokens in the root coordinator. If matched, the query will be sent to the next coordinator, and so on so forth, until it is accepted by a leaf coordinator and then forwarded to the data server specified by the outpointing link of the leaf coordinator. At any coordinator, if the input XPath step does not match the stored tokens, the query will be denied and dropped immediately.

3) Replication: Since all the queries are supposed to be processed first by the root coordinator, it becomes a single point of failure and a performance bottleneck. For robustness, we need to replicate the root coordinator as well as the coordinators at higher levels of the coordinator tree. Replication has been extensively studied in distributed systems. We adopt the passive path replication strategy to create the replicas for the coordinators along the paths in the coordinator tree, and let the *centralized authority* to create or revoke the replicas (please see more details in Section V). The CA maintains a set of replicas for each coordinator, where the number of replicas is either a preset value or dynamically adjusted based on the average queries passing through that coordinator.

4) Handling the predicates: In the original construction of NFA (similarly as described in QFilter [36] and QBroker [9]), a predicate table is attached to every child state of an NFA state as shown in Fig. 3. The predicate table stores predicate symbols (i.e., `pSymbol`), if any, in the corresponding query XPath step. An empty symbol ϕ means no predicate.

To handle the predicates, either from the query or from the ACR, the original strategy is *lookup-and-attach*. That is, if an XPath step in the query matches a child state in the state transition table (i.e., an `eSymbol`), predicate carried in that particular XPath step or predicate stored in the predicate table will be attached to the corresponding XPath step in the safe query. The real evaluation of the predicate is left to the data servers, which inevitably causes unnecessary communication and processing overhead if the predicate conditions conflict. We illustrate this problem with the following example.

Example 4. Consider a query $\mathsf{Q} = /\mathsf{site}/\mathsf{regions}//\mathsf{item}[@\mathsf{id} = "30"]/\mathsf{name}$ and the QBroker in Fig. 4. This query will be accepted at state 11, rewritten into a safe query, and sent to three data servers. However, from the index rules in Example 3, we know that data servers at 195.228.155.9 and 135.176.4.56 do not contain the requested data. Routing a query to irrelevant data servers yields unnecessary overhead.□

To address this problem, we present a new scheme to handle

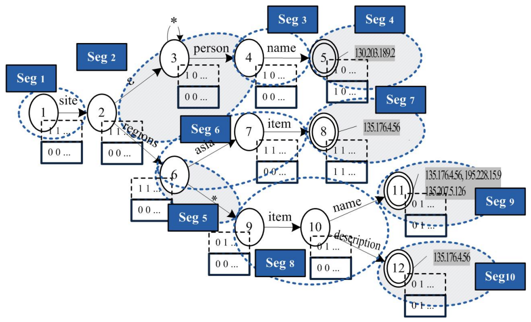
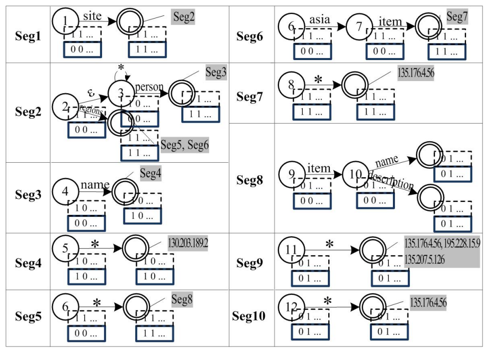



Fig. 5. An example to illustrate the automaton segmentation scheme: (a) divide the global automaton with granularity level of 1; (b) the segments are linked to form a tree structure.

value-based predicates in input XML queries. We first change the data structure of the original NFA state by adding new fields as condition, type, and location, to the predicate table: (1) pSymbol still stores the predicate token; (2) condition stores the test condition; (3) type $\in \{R, I\}$ indicates if the predicate is introduced from an ACR or an index rule, and (4) location stores the addresses carried by the index rule.

In processing, if the XPath step of a query does not have a predicate, the scheme works the same as before: it looks up the predicate table for predicates introduced by ACRs (i.e., with type $=$ R) and attaches "pSymbol||condition" to the safe query. If a predicate exists in a particular XPath step, the scheme retrieves records of the same predicate from the table, and sends them with the query predicate to a *predicate directory server*, which further examines the test conditions. In testing, the query predicate is first compared with the ACR predicate to determine if the query passes the access control testing. After that, the query predicate is further compared with the predicate introduced by the index rules, to limit the scope of potential destination data servers. If a predicate successfully passes both testings, we attach the location of the index rule to the safe query. Accordingly, in query processing, if an accepted query carries multiple location lists, it will be sent to the intersection of the destination data servers. We would like to point out that this scheme does not support structural predicates (that contain twig conditions in the predicates) due to the excessive overhead caused by waiting for responses from the twigs in a distributed setting. Last but not least, it is possible to employ searchable encryption approaches at the predicate directory servers, in the scenario where directory servers are not completed trusted. However, as we have discussed in Section II, several issues (e.g. computation) need to be addressed before searchable encryption could be employed; and their use is restricted to exact (and substring) matching for textual predicates.

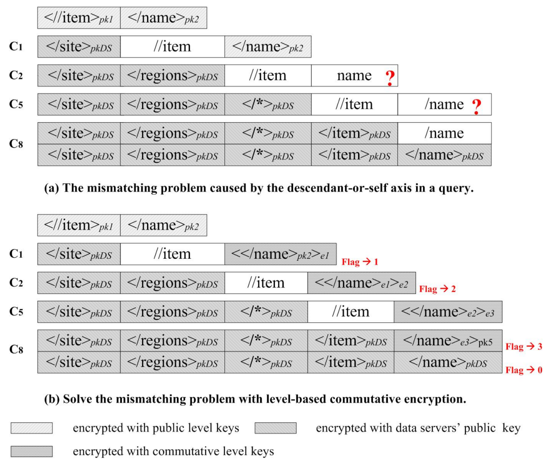
B. Query Segment Encryption

Informative hints can be learned from query content, so it is critical to hide the query from irrelevant brokering servers. However, in traditional brokering approaches, it is difficult, if not impossible, to do that, since brokering servers need to view query content to fulfill access control and query routing. Fortunately, the automaton segmentation scheme provides new opportunities to encrypt the query in pieces and only allows a coordinator to decrypt the pieces it is supposed to process. The query segment encryption scheme proposed in this work consists of the *pre-encryption* and *post-encryption* modules, and a special *commutative encryption* module for processing the double-slash ("//") XPath step in the query.

1) Level-based pre-encryption: According to the automaton segmentation scheme, query segments are processed by a set of coordinators along a path in the coordinator tree. A straightforward way is to encrypt each query segment with the public key of the coordinator specified by the scheme. Hence, each coordinator only sees a small portion of the query that is not enough for inference, but collaborating together, they can still fulfill the designed function. The key challenges in this approach is that the segment-coordinator association is unknown beforehand in the distributed setting, since no party other than the CA knows how the global automaton is segmented and distributed among the coordinators.

To tackle the problem, we propose to encapsulate query pieces based on the publicly known information – the global schema. XML schema also forms a tree structure, in which the *level* of a node in the schema tree is defined as its distance to the root node. Since both the ACR and index rules are constructed following the global schema, an XPath step (token) in the XPath expression of a rule is associated with level i if the corresponding node in the schema tree is at level i. We assume the nodes of the same level share a pair of public and private **level keys**, $\{pk, sk\}$. After automaton segmentation, the segments (and the corresponding coordinators) are assigned with the private key of level i, sk_i, if it contains a node of level i. In pre-encryption, the XPath steps (between two "/" or "//") of a query are encrypted with the public level keys $\{pk_1, pk_2, ...\}$, respectively. Intuitively, the ith XPath step of a query should be processed by a segment with a node at level i, and therefore, is able to be decrypted by the coordinator holding that segment. Moreover, if a coordinator has a segment that contains XML nodes of k different levels, it needs to decrypt the first k unprocessed XPath steps of the query.

2) Post-encryption: The processed query segments should also be protected from the remaining coordinators in later processing, so post-encryption is necessary. In a simple scheme, we assume all the data servers share a pair of public and private keys, $\{pk_{DS}, sk_{DS}\}$, where pk_{DS} is known to all the coordinators. Each coordinator first decrypts the query segment(s) with its private level key, performs authorization and indexing, and then encrypts the processed segment(s) with pk_{DS} so that only the data servers can view it.



<//item>pk1
</name>pk2
C1
</site>pkDS
//item
</name>pk2
C2
</regions>pkDS
name ?
C5
</*>pkDS
/name ?
C8
</item>pkDS
/name
</name>pkDS
(a) The mismatching problem caused by the descendant-or-self axis in a query.
<</name>pk2>e1
Flag → 1
<</name>e1>e2
Flag → 2
<</name>e2>e3
</name>e3>pk5
Flag → 3
Flag → 0
(b) Solve the mismatching problem with level-based commutative encryption.
encrypted with public level keys
encrypted with data servers' public key
encrypted with commutative level keys

Fig. 6. (a) An example of the mismatching problem; (b) An example of commutative encryption.

3) Commutative encryption for "//" handling: When a query has the *descendant-or-self* axis (i.e., "//" in XPath expressions), a so-called *mismatching problem* occurs at the coordinator who takes the "//" XPath step as input. This is because that the "//" XPath step may recursively accepts several tokens until it finds a match. Consequently, the coordinator with the private level key may not be the one that matches the "//" token, and vice versa. This problem is further explained in Example 5 and Fig. 6(a). To tackle the problem, we revise the level-based encryption scheme by adopting the commutative encryption. Commutative encryption algorithms [12], [54], [55] have the property of being *commutative*, where an encryption algorithm is commutative if for any two commutative keys e_1 and e_2 and a message m,$<< m >_{e_1} >_{e_2} = << m >_{e_2} >_{e_1}$. Therefore, we assign a new *commutative level key* e_i to nodes at level i, and further assume nodes at level i share e_i with nodes at level $i+2$.

Example 5. Assume the global automaton is segmented as shown in Fig. 5. Coordinators C_1, C_2, C_5, and C_8 hold the segments Seg_1, Seg_2, Seg_5, and Seg_8, respectively. According to the level-based pre-encryption, a query "//item/name" will be encrypted as $< //\mathsf{item} >_{\mathsf{pk}_1} < /\mathsf{name} >_{\mathsf{pk}_2}$. When C_1 gets "//item", it finds the token does not match the NFA state "site" but it still accepts the token due to the property of "//". Similarly, C_2, while still not be able to process "//item", will further decrypt irrelevant query segment "/name" with pk_2.□

The commutative encryption is invoked by the first coordinator encountering the "//" XPath step in a query and ended by the first coordinator whose NFA state matches the "//" token. The entire process experiences four stages. A flag $\in [0, 3]$ is attached to a query to indicate which the stage the query is currently at. Detailed algorithm is explained as follows:

1) When a coordinator C_m first encounters the "//" XPath step, it sets a pointer to the first unprocessed query segment, encrypts all the unprocessed query segments (except the "//" XPath step) with its commutative level key e_m, and sets the flag to 1.
2) With flag = 1, the next coordinator C_{m+1} first adds the

second encapsulation to the unprocessed query segments with its commutative level key e_{m+1}; then it decrypts the pointed query segment with its private level key sk_{m+1}, and moves the pointer to the next segment. After that, it sets the flag to 2.

3) For a following coordinator C_j, if none of its NFA states matches the "//" token, it first removes one wrapping (by decrypting with d_{j-2}) and adds one wrapping (by encrypting with e_j) to the unprocessed query segments. Due to the commutative encryption property, the unprocessed query segments are changed from $<< unprocessed >_{e_{j-2}}>_{e_{j-1}}$ to $<< unprocessed >_{e_{j-1}}>_{e_j}$. Then it decrypts the pointed segment with its private level key sk_j and moves the pointer to the next.
4) When a coordinator C_n accepts the "//" token, it applies the commutative decryption to all the unprocessed segments with key d_{n-2}, and encrypts each of them with the public level keys $\{pk_{n+1}, pk_{n+2}, ...\}$, respectively. After that, it sets the flag to 3.
5) Coordinator C_{n+1} decrypts all the unprocessed segments with d_{n-1} and resets the flag to 0.

The core idea of commutative encryption is to wrap the unprocessed query segments after the "//" XPath step with two consecutive commutative layer keys, which are not possessed by a same coordinator. The additional wrapping is kept until the commutative encryption process is stopped by a matching of the "//" token. In practical, we adopt Pohlig-Hellman exponentiation cipher with modulus p as our commutative encryption algorithm to generate the commutative keys.

Example 6. Let us revisit the previous example with commutative encryption. As shown in Fig. 6(b), once C_1 accepts the "//" XPath step "//item", it starts to add the commutative wrapping to the second query segment "/name" by encrypting with e_1. C_2 removes the wrapping of original level-based encryption and continues to add one more wrapping by encrypting wit e_2. C_5 still does not match the "//" token, so it only changes commutative wrapping to the ones related to e_2 and e_3. When C_8 matches '//item", it unwraps one layer of commutative encryption with d_2 and adds a level-based encryption with pk_5 to yield an output of $<< /\text{name} >_{e_3}>_{pk_5}$, which can be decrypted by C_9. □

C. The Overall PPIB Architecture

The architecture of PPIB is shown in Fig. 7, where users and data servers of multiple organizations are connected via a broker-coordinator overlay. In particular, the brokering process consists of four phases:

- **Phase 1:** To join the system, a user needs to authenticate himself to the local broker. After that, the user submits an XML query with each segment encrypted by the corresponding public level keys, and a unique session key K_Q. K_Q is encrypted with the public key of the data servers to encrypt the reply data.
- **Phase 2:** Besides authentication, the major task of the broker is metadata preparation: (1) it retrieves the role of the authenticated user to attach to the encrypted query; (2) it creates a unique Q_{ID} for each query, and attaches Q_{ID}, $< K_Q >_{pk_{DS}}$ and its own address to the query for data servers to return data.
- **Phase 3:** Upon receiving the encrypted query, the coordinators follow automata segmentation scheme and query segment encryption scheme to perform access control and query routing along the coordinator tree as described in Section IV-A and IV-B. At the leaf coordinator, all query segments should be processed and re-encrypted by the public key of the data server. If a query is denied access, a failure message with Q_{ID} will be returned to the broker.
- **Phase 4:** In the final phase, the data server receives a safe query in an encrypted form. After decryption, the data server evaluates the query and returns the data, encrypted by K_Q, to the broker that originates the query.

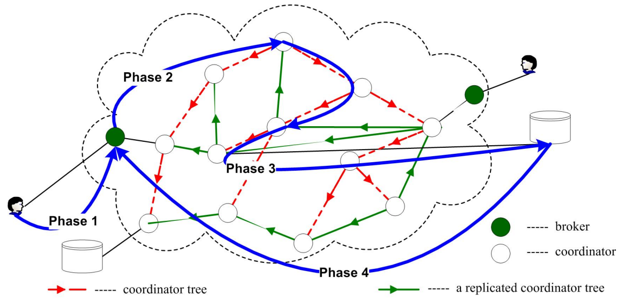


Fig. 7. We explain the query brokering process in four phases.

V. Maintenance

A. Key management

The CA is assumed for off-line initiation and maintenance. With the highest level of trust, the CA holds all the rules and plays a critical role in key management. There are four types of keys used in the brokering process: query session key K_Q, public/private level keys $\{pk, sk\}$, commutative level keys $\{e, d\}$, and public/private data server keys $\{pk_{DS}, sk_{DS}\}$. The query session keys created by the user, and the others are all generated and maintained by the CA. The data servers are treated as unique parties and share a pair of public and private keys, while each coordinator has its own pairs of level keys and commutative level keys. Along with the deployment of automaton segment, the CA creates key pairs for coordinators and assigns the private keys with the segments. The level keys need to be revoked in a batch once a certificate expires or when a coordinator at the same level quits the system.

B. Brokering Servers Join/Leave

Brokers and coordinators, contributed by different organizations, are allowed to dynamically join or leave the PPIB system. Besides authentication, a local broker only works as an entrance to the the coordinator overly. It stores the address of the root coordinator (and its replica) for forwarding the queries. When a new broker joins the system, it registers to the CA to receive the current address list from the CA and broadcasts its own address to the local users. When leaving the system, a broker only needs to broadcast a leave message to the local users. Thing are more complicate for the coordinators. Once joining the system, a new coordinator sends a join request to the CA. The CA authenticates its identity, and assigns automaton segments to it considering both the load balance requirement and its trust level. After that, the CA issues the corresponding private level keys and sends a broadcast $ServerJoin(addr)$ message to update the location list attached to the parent coordinator with the address of the newly joined coordinator. When a coordinator leaves the system, the CA decides whether to employ an existing or a new coordinator as a replacement, based on the heuristic rules for automaton deployment and the current load at each coordinator. After that, the CA broadcasts a $ServerLeave(addr_1, addr_2)$ message to replace the address of the old coordinator with the address of the new one in the location list at the dummy accept state of the parent coordinator. Finally, the CA revokes the corresponding level keys. If a failure is detected from a periodical status check by the CA or reported by a neighboring coordinator, the CA will treat the failed coordinator as a leaving server.

C. Metadata Update

ACR and index rules are updated to reflect the changes in the access control policy or the data distributions.

1) Index rules: To add or remove a (set of) data object, a local server sends an update message, in the form of $DataUpdate(object, address, action)$, to the CA, where *object* is an XPath expression, *address* is the location of the data object, and *action* is either "add" or "remove". To add a data object, the CA sends the update message to the root coordinator, from which the message traverses the coordinator network until reaching a leaf coordinator, where the $address$ will be added to its location list. A similar process is taken for data object removal to retrieve the corresponding leaf coordinators and remove the address from the location list.

2) Access control rules: For any change in the access control policy, we construct an $ACRUpdate(role, object, type)$ message to reflect the change for a particular $role$ and send it to the CA. The CA forwards the message to the root coordinator, from which the XPath expression in $object$ is processed by each coordinator according to its state transition table, in the same way as constructing an automaton with a new ACR: if the message stops at a particular NFA state, the state will be changed to an accept state for that role. Then, all the child and descendent leaf coordinators will be retrieved and the location lists will be attached to the accept state. If the message is accepted by an existing leaf coordinator, new automaton segments will be created and assigned to new coordinators. The location list at the original leaf coordinator will be copied to the new leaf coordinator.

VI. Privacy and Security Analysis

There are various types of attackers in the information brokering process. From their roles, we have abused insiders and malicious outsiders; from their capabilities, we have passive eavesdroppers and active attackers that can compromise any brokering server; from their cooperation mode, we have single and collusive attackers. In this section, we consider three most common types of attackers, local and global *eavesdroppers*, malicious *brokers* and malicious *coordinators*. We first analyze possible privacy breakages caused by each of them, and then summarize possible privacy exposures in Figure I.

3) Eavesdroppers: A local eavesdropper is an attacker who can observe all communication to and from the user side. Once an end user initiates an inquire or receives requested data, the local eavesdropper can seize the outgoing and incoming packets. However, it can only learn the location of local broker from the captured packets since the content is encrypted. Although local brokers are exposed to this kind of eavesdroppers, as a gateway of DIBS system, it prevents further probing of the entire DIBS. Although the disclosed broker location information can be used to launch DoS attack against local brokers, a backup broker and some recovery mechanisms can easily defend this type of attacks. As a conclusion, an external attacker who is not powerful enough to compromise brokering components is less harmful to system security and privacy.

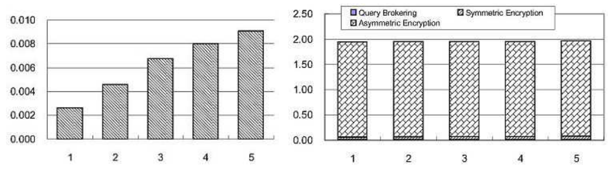
A global eavesdropper is an attacker who observes the traffic in the entire network. It watches brokers and coordinators gossip, so it is capable to infer the locations of local brokers and root-coordinators. This is because the assurance of the connections between user and broker, and between broker and root-coordinator. However, from the later-on communication, the eavesdropper cannot distinguish the coordinators and the data servers. Therefore, the major threat from a global eavesdropper is the disclosure of broker and root-coordinator location, which makes them targets of further DoS attack.

4) Single malicious broker: A malicious broker deviates from the prescribed protocol and discloses sensitive information. It is obvious that a corrupted broker endangers user location privacy but not the privacy of query content. Moreover, since the broker knows the root-coordinator locations, the threat is the disclosure of root-coordinator location and potential DoS attacks.

5) Collusive coordinators: Collusive coordinators deviate from the prescribed protocol and disclose sensitive information. Consider a set of collusive (corrupted) coordinators in the coordinator tree framework. Even though each coordinator can observe traffic on a path routed through it, nothing will be exposed to a single coordinator because (1) the sender viewable to it is always a brokering component; (2) the content of the query is incomplete due to query segment encryption; (3) the ACR and indexing information are also incomplete due to automaton segmentation; (4) the receiver viewable to it is likely to be another coordinator. However, privacy vulnerability exists if a coordinator makes reasonable inference from additional knowledge. For instance, if a leaf-coordinator knows how PPIB mechanism works, it can assure its identity (by checking the automaton it holds) and find out the destinations attached to this automaton are of some data servers. Another example is that one coordinator can compare the segment of ACR it holds with the open schemas and make reasonable inference about its position in the coordinator tree. However, inference made by one coordinator may be vague and even misleading.

Privacy type	**local eavesdropper**	**global eavesdropper**	**malicious broker**	**collusive coordinators**
User Location	Exposed	Exposed	Exposed	Protected
Query Content	Protected	Exposed	Exposed	Exposed only with compromised root coordinator
AC Policy	Protected	Protected	Protected	Exposed if path coordinators collude
Index Rules	Protected	Protected	Protected	Exposed if path coordinators collude
Data Distribution	Protected	Protected	Protected	Exposed if path coordinators collude
Data Location	Protected	Beyond suspicion	Protected	Exposed with malicious leaf coordinators

TABLE I
THE POSSIBLE PRIVACY EXPOSURE CAUSED BY FOUR TYPES OF ATTACKERS: LOCAL EAVESDROPPER (LE), GLOBAL EAVESDROPPER (GE), MALICIOUS BROKER (MB), AND COLLUSIVE COORDINATORS (CC).



(a) Average query brokering time at a coordinator. X: Number of keywords at a query broker; Y: Time (s)

(b) Average symmetric and asymmetric encryption time. X: Number of keywords at a query broker; Y: Time (ms)

Fig. 8. Estimate the overall processing time at each coordinator.

VII. Performance Analysis

In this section, we analyze the performance of proposed *PPIB* system using end-to-end query processing time and system scalability. In our experiments, coordinators are coded in Java (JDK 5.0) and results are collected from coordinators running on a Windows desktop (3.4G CPU). We use the XMark [56] XML document and DTD, which is wildly used in the research community. As a good imitation of real world applications, the XMark simulates an online auction scenario.

A. End-to-End Query Processing Time

End-to-end query processing time is defined as the time elapsed from the point when query arrives at the broker until to the point when safe answers are returned to the user. We consider the following four components: (1) average query brokering time at each broker/coordinator (T_C); (2) average network transmission latency between broker/coordinators (T_N); (3) average query evaluation time at data server(s) (T_E); and (4) average backward data transmission latency ($T_{backward}$). Query evaluation time highly depends on XML databases system, size of XML documents, and types of XML queries. Once these parameters are set in the experiments, T_E will remain the same (at seconds level [57]). Similarly, the same query set and ACR set will create the same safe query set, and the same data result will be generated by data servers. As a result, T_E and $T_{backward}$ are not affected by the broker-coordinator overlay network. We only need to calculate and compare the total forward query processing time ($T_{forward}$) as $T_{forward} = T_C \times N_{HOP} + T_N \times (N_{HOP} + 1)$. It is obvious that $T_{forward}$ is only affected by T_C, T_N, and the average number of hops in query brokering, N_{HOP}.

1) Average query processing time at the coordinator.: Query processing time at each broker/coordinator (T_C) consists of: (1) access control enforcement and locating next coordinator (Query brokering); (2) generating a key and encrypting the processed query segment (Symmetric encryption); and (3) encrypting the symmetric key with the public key created by super node (Asymmetric encryption).

To examine T_C, we manually generate 5 sets of access control rules, and partition the rules of each set into segments (keywords), which are assumed to be assigned to different coordinators in the following evaluation. From set 1 to set 5, the number of keywords held by one coordinator increases from 1 to 5. We also generate 1000 synthetic XPath queries and similarly divide the query into segments. In the experiment, we adopt the off-the-shelf cryptographic algorithms, 3DES for symmetric encryption and 1024-bit key length RSA (in practice, RSA-OAEP: RSA with optimal asymmetric encryption padding is recommended to defend against adaptive chosen-ciphertext attacks) for asymmetric encryption. Figure 8(a) shows that query brokering time is at milliseconds level, and increases linearly with the number of keywords at a site. As shown in Figure 8(b), since the data size is very small (the XPath token on average is 128 bits), encryption time for both symmetric and asymmetric encryption schemes is at milliseconds level, while the asymmetric encryption time dominates the total query processing time at each coordinator. As a result, average (T_C) is about *1.9 ms*. Query processing time at brokers and leaf-coordinators are shorter but still in the same level. For simplicity, we adopt the same value (i.e., *1.9 ms*) for the average query processing time at brokers and coordinators.

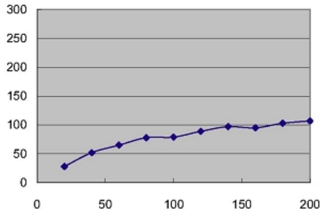
2) Average network transmission latency.: We adopt average Internet traffic latency *100 ms* as a reasonable estimation of T_N (from Internet traffic report) instead of using data collected from our gigabyte Ethernet.

3) Average number of hops.: We consider the case in which a query Q is accepted or rewritten by n ACRs $\{R_1, ..., R_n\}$ into the union of n safe sub-queries $\{Q'_1, ..., Q'_n\}$. When an accepted/rewritten sub-query Q'_i is processed by the rule R_i, the number of hops is determined by the number of segments of R_i. In the experiment, we generate a set of 200 synthetic access control rules and 1000 synthetic XPath queries.

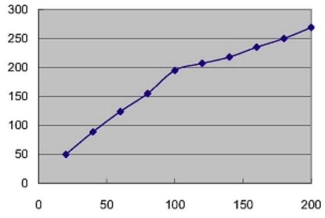
It is obvious to see that the more segments the global automaton is divided into, the more coordinators are needed and the less scalable the system is, due to the increased query processing cost. However, higher granularity leads to better privacy preserving performance. We choose the finest-granularity automaton segmentation (each XPath step of an ACR is partitioned as one segment and kept at one coordinator) for maximum privacy preserving. Our experiment result shows that N_{HOP} is 5.7, and the maximum number of hops of all queries is 8.

4) End-to-end query processing time.: From above experiment results, the total forward query processing time is calculated as $T_{forward} \simeq 1.9 \times 5.7 + 100 \times (5.7 + 1) \simeq 681(ms)$. It

X: number of access control rules; Y: number of coordinators.



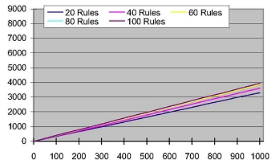
(a) Using simple path rules.



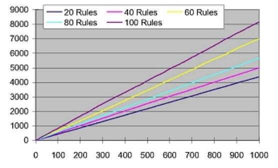
(b) Using XPath rules with wildcards.

Fig. 9. System scalability: number of coordinators.

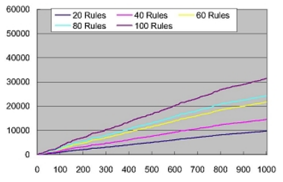
X: number of queries in a unit time; Y: number of total segments in the system.



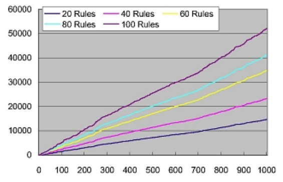
(a) Using simple XPath rules and simple XPath queries.



(b) Using simple XPath queries and rules with wildcards.



(c) Using queries and rules with 5% wildcards probability at each XPath step.



(d) Using query and ACR with 10% wildcards probability at each XPath step.

Fig. 10. System scalability: number of query segments.

is obvious that network latency $T_N * (N_{HOP} + 1)$ dominates total forward end-to-end query processing time, because the value of T_C is negligible compared with T_N. Moreover, since T_N remains the same (as an estimation from Internet traffic), N_{HOP} becomes the deterministic factor that affects end-to-end query processing time. Note that for other information brokering systems, although they use different query routing scheme, network latency is not avoidable. As a conclusion, the proposed PPIB approach achieves privacy-preserving query brokering and access control with limited computation.

B. System Scalability

We evaluate the scalability of the PPIB system against complicity of ACR, the number of user queries, and data size (number of data objects and data servers).

1) Complicity of XML schema and ACR.: When the segmentation scheme is determined, the demand of coordinators is determined by the number of ACR segments, which is linear with the number of access control rules. Assume finest granularity automaton segmentation is adopted, we can see that the increase of demanded number of coordinators is linear or even better, as shown in Figure 9(a) and (b). This is because similar access control rules with same prefix may share XPath steps, and save the number of coordinators. Moreover, different ACR segments (or, logical coordinators) may reside at the same physical site, thus reduce the actual demand of physical sites. In this framework, the number of coordinators, m, and the height of the coordinator tree, h, are highly dependent on how access control policies are segmented.

2) Number of queries: Considering n queries submitted into the system in a unit time, we use the total number of query segments being processed in the system to measure the system load. When a query is accepted as multiple sub-queries, all sub-queries are counted towards system load. For a query rejected after i steps, the first i segments are counted.

We generate 5 sets of synthetic ACRs and 10 sets of synthetic XML queries with different wildcard (i.e. "$/*$" and "$//$") probabilities at each XPath step. Figure 10 shows system load vs. number of XPath queries in a unit time. In particular, Figure 10(a) only has simple path rules (no wildcard or predicate), and Figure 10(b) has rules with wildcards. In both cases, system load increases linearly and average segments per query is less than 10. Figure 10(c) and (d) use the same set of ACRs as in Figure 10(b), but add wildcards into queries with probability 5% and 10% at each step, respectively. In the worst case, a query is processed no more than 50 segments. Moreover, we can see that larger ACR leads to higher system load, but the increase appears to be linear in all cases.

3) Data size: When data volume increases (e.g. adding more data items into the online auction database), the number of indexing rules also increases. This results in increasing of the number of leaf-coordinators. However, in PPIB, query indexing is implemented through hash tables, which is scalable. Thus, the system is scalable when data size increases.

VIII. Conclusion

With little attention drawn on privacy of user, data, and metadata during the design stage, existing information brokering systems suffer from a spectrum of vulnerabilities associated with user privacy, data privacy, and metadata privacy. In this paper, we propose PPIB, a new approach to preserve privacy in XML information brokering. Through an innovative automaton segmentation scheme, in-network access control, and query segment encryption, PPIB integrates security enforcement and query forwarding while providing comprehensive privacy protection. Our analysis shows that it is very resistant to privacy attacks. End-to-end query processing performance and system scalability are also evaluated and the results show that PPIB is efficient and scalable.

Many directions are ahead for future research. First, at present, site distribution and load balancing in PPIB are conducted in an ad-hoc manner. Our next step of research is to design an automatic scheme that does dynamic site distribution. Several factors can be considered in the scheme such as the workload at each peer, trust level of each peer, and privacy conflicts between automaton segments. Designing a scheme that can strike a balance among these factors is a challenge. Second, we would like to quantify the level of privacy protection achieved by PPIB. Finally, we plan to minimize (or even eliminate) the participation of the administrator node, who decides such issues as automaton segmentation granularity. A main goal is to make PPIB self-reconfigurable.

Acknowledgements

This work was partially supported by University of Kansas General Research Fund (GRF), NRGRF 2302283, NSF OIA-1028098, AFOSR FA9550-07-1-0527 (MURI), ARO W911NF-09-1-0525 (MURI), NSF CNS-0905131, NSF CNS-0916469, ARO W911NF1210055, and U.S. ARL and U. K. MoD W911NF-06-3-0001.

References

[1] W. Bartschat, J. Burrington-Brown, S. Carey, J. Chen, S. Deming, and S. Durkin, "Surveying the RHIO landscape: A description of current RHIO models, with a focus on patient identification," *Journal of AHIMA 77*, pp. 64A–D, January 2006.

[2] A. P. Sheth and J. A. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," *ACM Computing Surveys (CSUR)*, vol. 22, no. 3, pp. 183–236, 1990.

[3] L. M. Haas, E. T. Lin, and M. A. Roth, "Data integration through database federation," *IBM Syst. J.*, vol. 41, no. 4, pp. 578–596, 2002.

[4] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming," in *Proceedings of IEEE INFOCOM*, 2005.

[5] A. C. Snoeren, K. Conley, and D. K. Gifford, "Mesh-based content routing using XML," in *SOSP*, pp. 160–173, 2001.

[6] N. Koudas, M. Rabinovich, D. Srivastava, and T. Yu, "Routing XML queries," in *ICDE '04*, p. 844, 2004.

[7] G. Koloniari and E. Pitoura, "Peer-to-peer management of XML data: issues and research challenges," *SIGMOD Rec.*, vol. 34, no. 2, 2005.

[8] M. Franklin, A. Halevy, and D. Maier, "From databases to dataspaces: a new abstraction for information management," *SIGMOD Rec.*, vol. 34, no. 4, pp. 27–33, 2005.

[9] F. Li, B. Luo, P. Liu, D. Lee, P. Mitra, W. Lee, and C. Chu, "In-broker access control: Towards efficient end-to-end performance of information brokerage systems," in *Proc. IEEE SUTC*, 2006.

[10] F. Li, B. Luo, P. Liu, D. Lee, and C.-H. Chu, "Automaton segmentation: A new approach to preserve privacy in XML information brokering," in *ACM CCS '07*, pp. 508–518, 2007.

[11] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, 1981.

[12] R. Agrawal, A. Evfimivski, and R. Srikant, "Information sharing across private databases," in *Proceedings of the 2003 ACM SIGMOD*, 2003.

[13] M. Genesereth, A. Keller, and O. Duschka, "Informaster: An information integration system," in *SIGMOD*, (Tucson), 1997.

[14] I. Manolescu, D. Florescu, and D. Kossmann, "Answering XML queries on heterogeneous data sources," in *VLDB*, pp. 241–250, 2001.

[15] J. Kang and J. F. Naughton, "On schema matching with opaque column names and data values," in *SIGMOD*, pp. 205–216, 2003.

[16] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for Internet applications," in *IEEE/ACM Transactions on Networking*, vol. 11 of *1*, 2003.

[17] R. Huebsch, B. Chun, J. Hellerstein, B. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. Yumerefendi, "The architecture of PIER: an Internet-scale query processor," in *CIDR*, pp. 28–43, 2005.

[18] O. Sahin, A. Gupta, D. Agrawal, and A. E. Abbadi, "A peer-to-peer framework for caching range queries," in *ICDE*, 2004.

[19] A. Carzaniga, M. J. Rutherford, and A. L. Wolf, "A routing scheme for content-based networking," in *Proc. of INFOCOM*, 2004.

[20] Y. Diao, S. Rizvi, and M. J. Franklin, "Towards an Internet-scale XML dissemination service," in *VLDB Conference*, (Toronto), August 2004.

[21] G. Koloniari and E. Pitoura, "Content-based routing of path queries in peer-to-peer systems.," in *EDBT*, pp. 29–47, 2004.

[22] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *IEEE Symposium on Security and Privacy*, pp. 44 –55, 2000.

[23] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *ICDCS'10*.

[24] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *CRYPTO'07*.

[25] M. Li, S. Yu, N. Cao, and W. Lou, "Authorized private keyword search over encrypted data in cloud computing," in *ICDCS'11*, pp. 383 –392, june 2011.

[26] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *TCC'07*.

[27] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC'09*.

[28] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, "Keyword search and oblivious pseudorandom functions," in *TCC'05*.

[29] M. K. Reiter and A. D. Rubin, "Crowds: anonymity for Web transactions," *ACM TISS*, vol. 1, no. 1, pp. 66–92, 1998.

[30] P. F. Syverson, D. M. Goldschlag, and M. G. Reed, "Anonymous connections and onion routing," in *IEEE Symposium on Security and Privacy*, (Oakland, California), pp. 44–54, 4–7 1997.

[31] W. Tolone, G.-J. Ahn, T. Pai, and S.-P. Hong, "Access control in collaborative systems," *ACM Comput. Surv.*, vol. 37, no. 1, 2005.

[32] S. Cho, S. Amer-Yahia, L. V. S. Lakshmanan, and D. Srivastava, "Optimizing the secure evaluation of twig queries.," in *VLDB*, 2002.

[33] M. Murata, A. Tozawa, and M. Kudo, "XML access control using static analysis," in *ACM CCS*, 2003.

[34] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy, "Extending query rewriting techniques for fine-grained access control," in *SIGMOD'04*, (Paris, France), pp. 551–562, 2004.

[35] T. Yu, D. Srivastava, L. V. S. Lakshmanan, and H. V. Jagadish, "Compressed accessibility map: Efficient access control for XML," in *VLDB*, (China), pp. 478–489, 2002.

[36] B. Luo, D. Lee, W. C. Lee, and P. Liu, "Qfilter: Fine-grained run-time XML access control via nfa-based query rewriting enforcement mechanisms," in *CIKM*, 2004.

[37] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernndez, M. Kay, J. Robie, and J. Simon, "XML path language (XPath) version 2.0." `http://www.w3.org/TR/xpath20/`, 2003.

[38] E. Damiani, S. Vimercati, S. Paraboschi, and P. Samarati, "A fine-grained access control system for XML documents," *ACM TISSEC*, vol. 5, no. 2, pp. 169–202, 2002.

[39] E. Damiani, S. di Vimercati, S. Paraboschi, and P. Samarati, "Securing XML documents," *EDBT 2000*, pp. 121–135, 2000.

[40] H. Zhang, N. Zhang, K. Salem, and D. Zhuo, "Compact access control labeling for efficient secure XML query evaluation," *Data & Knowledge Engineering*, vol. 60, no. 2, pp. 326–344, 2007.

[41] Y. Xiao, B. Luo, and D. Lee, "Security-conscious XML indexing," *Advances in Databases: Concepts, Systems and Applications*, 2007.

[42] E. Bertino, S. Castano, and E. Ferrari, "Securing XML Documents with AuthorX," *IEEE Internet Computing*, vol. 5, no. 3, pp. 21–31, 2001.

[43] E. Damiani, S. Vimercati, S. Paraboschi, and P. Samarati, "Design and implementation of an access control processor for XML documents.," *Computer Networks*, vol. 33, no. 1-6, pp. 59–75, 2000.

[44] A. Gabillon and E. Bruno, "Regulating access to xml documents," in *Proc. DAS*, pp. 299–314, 2002.

[45] W. Fan, C.-Y. Chan, and M. Garofalakis, "Secure xml querying with security views," in *ACM SIGMOD*, pp. 587–598, 2004.

[46] M. Kudo, "Access-condition-table-driven access control for XML databases," *ESORICS 2004*, pp. 17–32, 2004.

[47] S. Mohan, A. Sengupta, and Y. Wu, "Access control for XML: a dynamic query rewriting approach," in *Proc. IKM*, pp. 251–252, 2005.

[48] N. Qi and M. Kudo, "XML access control with policy matching tree," *ESORICS 2005*, pp. 3–23, 2005.

[49] L. Bouganim, F. D. Ngoc, and P. Pucheral, "Client-based access control management for XML documents.," in *VLDB*, pp. 84–95, 2004.

[50] S. Abiteboul, A. Bonifati, G. Cobéna, I. Manolescu, and T. Milo, "Dynamic xml documents with distribution and replication," in *ACM SIGMOD*, pp. 527–538, ACM, 2003.

[51] P. Skyvalidas, E. Pitoura, and V. Dimakopoulos, "Replication routing indexes for xml documents," in *DBISP2P Workshop*, 2007.

[52] G. Skobeltsyn, *Query-driven indexing in large-scale distributed systems*. PhD thesis, EPFL, 2009.

[53] P. Rao and B. Moon, "Locating xml documents in a peer-to-peer network using distributed hash tables," *TKDE*, vol. 21, no. 12, 2009.

[54] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Zhu, "Tools for privacy preserving distributed data mining," *ACM SIGKDD Explorations*, vol. 4, no. 2, 2003.

[55] H. Y. S. Lu, "Commutative cipher based en-route filtering in wireless sensor networks," in *VTC*, vol. 2, pp. 1223– 1227, Sept. 2004.

[56] A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu, and R. Busse, "XMark: a benchmark for XML data management," in *VLDB*, pp. 974–985, 2002.

[57] H. Lu, J. X. Yu, G. Wang, S. Zheng, H. Jiang, G. Yu, and A. Zhou, "What makes the differences: benchmarking xml database implementations," *ACM Trans. Inter. Tech.*, vol. 5, no. 1, pp. 154–194, 2005.