The VLDB Journal
DOI 10.1007/s00778-008-0129-7

REGULAR PAPER

Privacy-preserving indexing of documents on the network

Mayank Bawa · Roberto J. Bayardo Jr · Rakesh Agrawal · Jaideep Vaidya

Received: 15 December 2007 / Revised: 15 November 2008 / Accepted: 20 November 2008

Abstract With the ubiquitous collection of data and creation of large distributed repositories, enabling search over this data while respecting access control is critical. A related problem is that of ensuring privacy of the content owners while still maintaining an efficient index of distributed content. We address the problem of providing privacy-preserving search over distributed access-controlled content. Indexed documents can be easily reconstructed from conventional (inverted) indexes used in search. Currently, the need to avoid breaches of access-control through the index requires the index hosting site to be fully secured and trusted by all participating content providers. This level of trust is impractical in the increasingly common case where multiple competing organizations or individuals wish to *selectively* share content. We propose a solution that eliminates the need of such a trusted authority. The solution builds a centralized *privacy-preserving* index in conjunction with a distributed *access-control enforcing* search protocol. Two alternative methods to build the centralized index are proposed, allowing trade offs of efficiency and security. The new index provides strong and quantifiable privacy guarantees that hold *even* if the entire index is made public. Experiments on a real-life dataset validate performance of the scheme. The appeal of our solution is twofold: (a) content providers maintain complete control in defining access groups and ensuring its compliance, and (b) system implementors retain tunable knobs to balance privacy and efficiency concerns for their particular domains.

Keywords Privacy · Indexing · Distributed search

Dr. Vaidya's work was supported by the National Science Foundation under grant CNS-0746943 and by a research resources grant from Rutgers Business School, Newark and New Brunswick.

M. Bawa
Aster Data Systems, Redwood City, CA 94065, USA
e-mail: bawa@db.stanford.edu

R. J. Bayardo Jr
Google, Inc., Mountain View, CA 94043, USA
e-mail: roberto.bayardo@gmail.com

R. Agrawal
Microsoft Search Labs, Mountain View, CA 94043, USA
e-mail: rakesh.agrawal@microsoft.com

J. Vaidya (✉)
Rutgers University, Newark, NJ 07102, USA
e-mail: jsvaidya@business.rutgers.edu

1 Introduction

While private and semi-private information on the network has grown rapidly in recent years, mechanisms for searching this information have failed to keep pace. A user faced with the problem of locating an access-controlled document must typically identify and search each relevant repository, assuming of course the user knows and remembers which repositories are relevant!

The lack of tools for searching access-controlled content on the network stems from the considerable difficulty in creating a search-engine that indexes the content while respecting the security and privacy requirements of the content providers. Contemporary search engines [8,26,31] build inverted indexes that map a keyword to its precise locations in an indexed document. The indexed document can thus be easily reconstructed from the index. Conferred with knowledge of *every* searchable document, the trust required of a search engine over access-controlled content grows rapidly with each participating provider. This enormous trust requirement, coupled with the potential for a complete breach of access control by way of malicious index disclosure, render such an approach impractical.

In this paper we address the problem of providing an efficient search mechanism that respects privacy concerns of the participating content providers. Our solution is to build a centralized index of content that works in conjunction with an *access control enforcing* search protocol across networked providers. The centralized index itself provides strong and quantifiable privacy guarantees that hold *even* if the entire index is made public. The degree of privacy provided by the index can to be tuned to fit the needs of the providers, and overhead incurred by the search protocol is proportional to the degree of privacy provided. For example, consider an organization such as Google or Microsoft which may want to index the access controlled repositories of several document providers (such as patent providers). Due to privacy concerns the providers may not want to reveal their list of patents directly to any third party without appropriate credentials. In this case, all of the providers could collaborate to create a privacy preserving index which is kept at Google/Microsoft. Queriers can use the Google/Microsoft search service to find providers who may have the documents they want, and then directly contact and negotiate with the providers to gain access to the documents. For the rest of this paper, we will assume this as our expository example with around eight document providers having their own content, and go through the different algorithm steps.

We envision applications of this technology in various sectors, where multiple organizations are actively competing as well as collaborating with constantly evolving alliances. Another application domain is file-sharing through personal webservers (e.g., YouServ [3]). For example, our scheme could be used by individuals to share copyrighted songs electronically with others who can authenticate they already own the song. The providers can keep track of the proofs supplied to allow audit of such exchanges.

Our method of providing efficient search over access-controlled content preserves the important appeal of private information sharing—each provider has complete control over the information it shares: how much is shared, when it is shared, and with whom it is shared. Index updates can be easily handled—simply by rerunning the basic protocols to create the global index. However, more efficient solutions are possible, and need to be investigated.

The key contribution of this paper is to propose this notion of a privacy-preserving search index, along with developing two alternative methods for the construction of such an index. The randomized construction technique is very efficient but has lower security, and requires certain trust assumptions. The cryptographic technique is much slower but has significantly stronger security properties. It is also resistant to collusion and makes no trust assumptions. The choice of which technique to use can be made on the basis of the domain requirements and acceptable tradeoffs between privacy and efficiency. Experiments on a real life dataset validate the overall performance of the technique.

1.1 Organization

The rest of the paper is organized as follows. Section 2 presents the preliminaries, including the problem statement, infrastructure assumptions, privacy goals as well as an adversary model. Section 3 analyzes conventional search solutions and presents possible privacy attacks against these. Section 4 defines a privacy-preserving index structure that can protect against such attacks. Sections 5 and 6 provide two alternative mechanisms (randomized and cryptographic) for constructing such an index. Section 7 evaluates both construction methods as well as the performance of the index on a real-life dataset. Section 8 presents the related work in the literature. Finally, Sect. 9 concludes the paper and discusses avenues for further research.

2 Preliminaries

In this section, we define the problem of searching distributed access-controlled content and the assumptions our solution makes on the supporting infrastructure. We also present the privacy goals that are the focus of this paper, followed by a privacy spectrum for characterizing the degree with which any solution achieves them.

2.1 Problem statement

The input to the problem of searching distributed access-controlled content is a set of *content providers* $p_1, p_2, \ldots, p_n$, and a *searcher* s who issues a *query* q. Each provider is said to *share* a set of documents with access-control determined by the authenticated identity of the searcher s and an *access policy*. The desired output is the set containing documents d such that (1) d is shared by some provider p_i for $1 \leq i \leq n$, (2) d matches the query q and (3) d is accessible to s as dictated by p_i's access policy.

2.2 Assumptions on the infrastructure

Most of the details of the query language, access policy language, and authentication mechanism are immaterial to our approach. We require only the following properties of each component:

A *Query language*: The query language must support conjunctive keyword queries. Additional constructs (e.g., phrase search, negated terms) can be supported as well, so long as they only further constrain the result set.

B *Authentication mechanism*: The authentication scheme should allow users to authenticate themselves to each content provider independently, preferably without requiring explicit registration with each provider. For example, client authentication through third-party signed security certificates (e.g., SSL/TLS [13,19]) would be satisfactory.

C *Access policy language*: The only requirement of the access policy language is that content providers are themselves able to apply and enforce their access policies given the authenticated identity of the searcher. This allows, for example, each content provider to individually select a policy language that best fits its requirements.

2.3 Privacy adversaries

Just as important as ensuring correct output for a query q is the requirement of preventing an *adversary* from learning what one or more providers may be sharing without obtaining proper access rights. We will characterize solutions to the problem in terms of their susceptibility to privacy breaches by the types of adversaries described here.

A *passive adversary* is an eavesdropper who merely observes and records messages (queries, responses, indexes) sent in the system. Such an adversary may have either a *global* (ability to observe all messages in the system) or a *local* (ability to observe messages sent to/from a particular content provider) view of the system. An *active adversary* is an entity which acts with deliberate intent in accordance with the system protocol to gather information. In our model, such an adversary could inspect index structures, issue various queries, or even participate in the index construction process to facilitate such breaches. Adversaries may also *collude* with each other.

Adversaries may also be categorized according to roles they can assume. For example, most users (and hence adversaries) will be limited to performing the role of a searcher since content providers are in practice likely to be a smaller and more controlled population. The information and operations accessible through each role (searcher, provider, indexer) can be used to facilitate different types of breaches.

2.4 Privacy goal

Privacy goal We focus on attaining the following privacy goal with respect to a document d made searchable by some content provider p:

Content privacy *An adversary A should not be allowed to deduce that p is sharing some document d containing keywords q unless A has been granted access to d by p.*

Other privacy goals related to distributed search but not addressed in this paper include *query privacy* and *provider anonymity*. Query privacy involves preventing an adversary from determining which searcher issued what particular queries. Provider anonymity involves preventing an adversary from determining which provider published what particular documents.

2.5 Degrees of privacy

To formally analyze a privacy-preserving scheme, we need to characterize the degree with which Content Privacy is attained against an adversary that does not have access to a document d being shared by provider p. To this end, we adapt the privacy spectrum used by Reiter and Rubin in their analysis of Crowds [33] as shown in Fig. 1 and discussed below:

A *Provable exposure*: The adversary can provide irrefutable evidence that p is sharing d.

B *Possible innocence*: The claim of adversary about p sharing d can be false with a non-trivial probability (e.g., with probability in (0.5, 1)).

C *Probable innocence*: The claim of adversary about p sharing d is more likely to be false than true (e.g., with probability in (0, 0.5]).

D *Absolute privacy*: The adversary cannot determine if p is sharing d or not.

E *Beyond suspicion*: The adversary cannot determine if p is more likely to be sharing document d than any other provider.

We can replace d in the above discussion by any set of keywords q, in which case our aim is to prevent the adversary from determining whether p is sharing a document that contains q.

3 Analysis of conventional solutions

In this section, we consider search solutions adopted by conventional systems and how they might be adapted to support search over access-controlled content. Such adaptations fail to address our privacy and efficiency goals, but their analysis provides insight into designing a search mechanism.

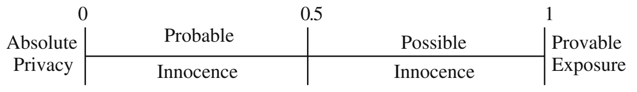

Fig. 1 Privacy on a probabilistic scale of 0 (*Absolute privacy*) to 1 (*Provable exposure*)

3.1 Centralized indexing

The most common scheme for supporting efficient search over distributed content is to build a centralized inverted index. The index maps each term to a set of documents that contain the term. The index is queried by the searcher to obtain a list of matching documents. This is the scheme of choice of web search engines [8], mediators [26], and the now defunct Napster [31] network. The scheme can be extended to support access-controlled search by propagating access policies along with content to the indexing host. The index host must apply these policies for each searcher to filter search results appropriately. Since only the indexing host needs to be contacted to completely execute a search, searches are highly efficient.
Privacy breaches: A centralized index will Provably Expose content providers to anyone who has access to the index structure. In cases where the index host is completely trusted by all content providers, this violation of access control may be tolerable. Unfortunately, finding such a trusted host is immensely difficult. Worse, compromise of the index host by hackers could lead to a complete and devastating privacy loss should the index be revealed publicly.

3.2 Query flooding

At the other end of the efficiency spectrum lie flood-based schemes that send the query to *all* participating content providers. Such schemes include the Gnutella [21] network, where providers locally evaluate each query and directly provide any matching documents to the searcher. We can think of an augmented Gnutella-based search protocol that implements access control. In such a protocol, the query will be flood along with the identity and IP address of the query originator. Providers could securely deliver search results back to the authenticated searcher over an encrypted connection [19]. Since content shared by a provider p resides at p alone, providers are assured Absolute Privacy and the goal of Content Privacy is preserved.
Performance limitations: While the above adaptation has excellent privacy characteristics, flooding suffers from poor scalability and severe performance penalties. The protocols hence adopt heuristics (e.g., time-to-live fields) that limit search horizons and compromise completeness.

3.3 Distributed indexing

The performance limitations of query broadcasting have led to work on distributed indexing methods that support efficient search without the need for a single centralized index provider. KaZaa [28], for example, is a P2P network that leverages "super-peers" (machines with above-average bandwidth and processing power) by having them host sub-indexes of content shared by several less capable machines.

The distributed index is used to identify a set of documents (or hosts of documents) matching the searcher's query. These hosts are then contacted directly by the searcher to retrieve the matching documents. Access control can be supported by simply having the providers enforce their access policies before providing the documents.
Privacy breaches: Much as in the case of a centralized index, any node with access to a portion of the distributed index can Provably Expose any of the providers indexed by that portion. Worse, indexes are hosted by untrusted machines over whom the providers themselves have no control. An active adversary that does not host a portion of the index can search the distributed index to inflict privacy breaches. For example, the adversary can determine the precise list of providers sharing a document with a particular keyword by issuing a search on that keyword—a breach of Content Privacy with Provable Exposure. Content Privacy can also be breached by mounting *phrase attacks*. Such attacks take advantage of the observation that most documents have characteristic sets of words that are unique to them [10]. To identify a provider sharing some document, the adversary need only compose a query consisting of such terms for the document. The resulting list of sites are then known to share the document but with Possible Innocence. By choosing an appropriate set of terms, the adversary can achieve a near Provable Exposure.

3.4 Centralized fuzzy indexing

Some search applications do not maintain precise inverted index lists, but instead maintain structures that allow mapping of a query to a "fuzzy" set of providers that *may* contain matching documents. For example, YouSearch [2] builds a centralized Bloom filter [6] index. The Bloom filter index can be probed by a searcher to identify a list of all providers that contain documents matching the query. The list however is not necessarily precise, since bloom filters may produce *false positives* due to hash collisions. Given such a list, the searcher must contact each provider to accumulate results. These schemes can be extended to support access-controlled search by having the providers enforce their access policies at the point a searcher requests matching documents.
Privacy breaches: Bloom filter indexes do offer limited privacy characteristics by virtue of potential false positives in the list of providers. Each provider in the list is thus Possibly Innocent of sharing a document matching the query. However, this privacy is spurious. An active adversary can perform a *dictionary-based attack* on the Bloom filter index to identify the term distribution of any indexed provider. Dictionary-based attacks take advantage of the fact that sentences in natural language (e.g., English) use words from a restricted vocabulary that are easily compiled (e.g., in a

Oxford/Webster dictionary). Thus, the adversary can compute a hash for each word in the vocabulary. A provider in the Bloom filter entry for such a hash is, with some probability, sharing a document with the corresponding word. In addition, the scheme remains prone to phrase attacks.

4 A privacy-preserving index (PPI)

Any search mechanism that relies on a conventional search index allows a provider to be Provably Exposed because of the precise information the index itself conveys. Efficient privacy-preserving search therefore requires an index structure that prevents Content Privacy breaches even in the event that the index is made public. In this section, we define such an index structure and analyze its privacy characteristics. We show that any index satisfying our definition leaves providers with at least Probable Innocence in response to active adversary attacks on the index structure. Section 5 presents two algorithms for constructing such an index.

4.1 Search methodology

While a conventional inverted list maps queries to lists of matching documents, an index that preserves privacy maps queries to lists of matching *providers*. Given the list of providers that may satisfy a query, it is then up to the searcher to directly query such providers and request matching documents. The providers, on receiving a query and authenticating the searcher, return a list of documents filtered according to the searcher's access rights.

By implementing search in this manner, we have moved the point of control from the index-hosting site to the providers. Providers can now manage and enforce access policies themselves without relying on any central host. While there is an efficiency penalty associated with the need to individually contact the providers, experimental results over publicly shared content [2] indicate the performance of such an approach can be quite reasonable in practice, even when there are many (>1,500) providers.[1]

4.2 Definition

A PPI is a mapping function built on the set of documents D being shared by the set of providers $p_1, p_2, \ldots, p_n$. It accepts a query q and returns a subset of providers M that may contain matching documents. In order for the function to be considered *privacy preserving*, the set M for any query q must satisfy one of the following conditions:

[1] Organizations that share many documents could participate using multiple *virtual* providers where each virtual provider is responsible for handling search requests for a specific sub-repository.

A M is the *null set* only if there is no document in D that matches q.
B M is a *subset of providers* consisting of all providers that share a document matching q ("true positives") and an equal or greater number of providers that do not share a matching document ("false positives").
C M is the set of *all providers*.

The PPI must behave like a conventional index: over time the index must return identical results for identical queries unless indexed content itself has changed. In addition, for any query q' whose results are a subset of another query q, the result set returned for q' must be a subset of that returned for q. These behavioral requirements prevent attacks that attempt privacy breaches by filtering out of false positives.

The PPI must be implemented with care: a naive implementation could easily yield more information than is provided by the PPI definition. For example, the indexing host might aggregate all shared content locally and preprocess it to materialize an index with true positives alone; the false positives as required by the definition being inserted into results at query time. Notice that in this case the materialized index itself does not correspond to PPI definitions. A public disclosure of the materialized index would result in Provable Exposure of content providers. Instead, we require that a materialized index should not yield any more information than that obtained from executing an exhaustive list of queries against the PPI.

4.3 Correctness

The set M returned by PPI for a query q never excludes any true positives for q. In other words, the result set for q will contain all providers that have at least one matching document. The searcher contacts each provider to accumulate the results, who will release a document if and only if the searcher is allowed to access it. Thus, searching with a PPI leads to correct output.

4.4 Privacy characteristics

Recall that the adversary who inspects the index has no advantage over the adversary issuing queries at will other than the time required for exploring the space of all queries. We can therefore restrict our analysis to the latter case.

Results for any query the adversary issues can correspond to one of Cases [A], [B] or [C] as defined above. If the result corresponds to Case [A], the adversary learns that no provider offers any document containing the specific term. All providers are Beyond Suspicion in that none is known to be more likely than the others to share such documents.

If the result corresponds to Case [B], at least half of the set of identified providers are false positives. Thus, all true positives within the set have Probable Innocence with respect to actually sharing a matching document. All providers outside the identified set are Beyond Suspicion.

If the result corresponds to Case [C], the adversary is unable to discriminate between providers. In effect, the index has degenerated into a broadcast-based mechanism, where all providers are Beyond Suspicion of sharing matching documents.

To relate privacy characteristics attained by the index with our goal of Content Privacy, we claim that by ensuring providers are always at least Probably Innocent for any inspection of the index by an active adversary, the adversary cannot bring about a strong privacy breach by exploiting a PPI alone. Note also that collusion between adversarial searchers offers no additional opportunities for privacy breaches.

One limitation of our solution is that it allows an adversary to determine which group has a document even though he may not have the rights to access the document, which is a leakage of information. The adversary may use this information along with his other resources for malicious purposes. This may be avoided by demanding that a querier exhibit their credentials allowing them to search for a particular document, or to search a group whose members it would have access to, *before* executing the search. However, this would make the search process significantly more expensive, and it does not fit the standard search environment, which assumes that you can search without providing your credentials. Nevertheless, being aware of this leakage is important, though being able to handle it in an efficient manner is outside the scope of this paper.

4.5 Efficiency

Define *selectivity* σ of a query q to be the fraction of providers that share a document matching q. Observe that Case [B] causes the PPI to be at least $2\times$ less selective than an index which precisely maps queries to providers. Also observe that an optimally-efficient PPI must use Case [C] minimally: only for queries that have a selectivity $\sigma > 0.5$ that precludes them from Case [A] (trivially) and Case [B] (absence of an equal number of false positives). Hence, the PPI need not be more than $2\times$ less selective than a precise inverted index. Note however that there is an inherent trade-off between efficiency and the degree of Probable Innocence offered by a PPI. A PPI with optimal efficiency can never offer more than 50% false positives for queries in Case [B].

An optimally efficient PPI yields the absolute minimum level of privacy (an adversary's claim is false with probability 0.5) required to satisfy the definition of Probable Innocence. Such a low degree of Probable Innocence may be inadequate for highly sensitive domains. Implementations should thus offer a means by which the level of false positives can be increased (at the expense of efficiency) to achieve a desired privacy level for the application domain. We discuss such an implementation scheme next.

5 Constructing a randomized PPI

A procedure for constructing a PPI must address not only the correctness of the resulting structure, but also the potential for privacy breaches during the construction process. Ensuring privacy in the presence of adversarial participants is non-trivial since the index construction process involves pooling together information about content shared by each provider.

We now present a randomized procedure to construct an index that is expected to be privacy-preserving for any particular query. The procedure partitions providers into "privacy groups" of size c. Within a group, providers are arranged in a ring. The providers execute a randomized algorithm which has only a small probability of error. We quantify this probability of error and show that by tuning a parameter, the error can be made small enough to be irrelevant. We show that the construction process ensures that providers are resilient to breaches beyond Probable Innocence.

There are two exceptions where a provider may suffer a breach larger than Probable Innocence from adversaries *within* its privacy group. Providers who immediately precede an active adversary will be assured of only Possible Innocence with respect to sharing documents with a particular term. Specifically, an adversary neighbor can determine whether its predecessor along the ring is sharing a specific term with *at best* 0.71 probability.

The second exception is for a provider when both its neighbors along the ring collude against it. In such a case, the provider can be Provably Exposed as sharing documents containing particular terms. We argue that such a breach can be minimized by having providers choose their two neighbors on the ring based on previously established real-world trust relationships. However, when this is not possible, technical solutions are desirable to solve the problem.

Section 6 presents such a technical solution—a deterministic secure procedure for generating a privacy-preserving index. This procedure eliminates the two exceptional breaches discussed above. However, the secure approach is less efficient than the randomized procedure. The choice of procedures can be made based on the security and efficiency demands of a particular domain. For the rest of this section and the next, we use the expository example presented in the introduction to help clarify how the different algorithms work and how the privacy preserving index can be constructed. In this example, we assume that there are eight providers that would engage in creating the index.

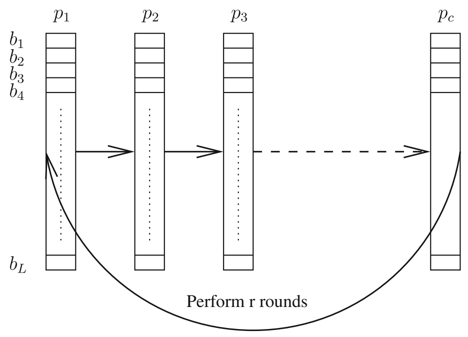

Fig. 2 Index construction proceeds over r rounds

```
IndexConstruction($r, V_s, V'_G$)
    $P_{ex} := 1/2^r$
    $P_{in} := 1 - P_{ex}$
    for ($i := 1; i < L; i := i + 1$)
        do
            if ($V_s[i] = 1$ and $V'_G[i] = 0$)
            then Set $V'_G[i] := 1$ with prob. $P_{in}$
            if ($V_s[i] = 0$ and $V'_G[i] = 1$)
            then Set $V'_G[i] := 0$ with prob. $P_{ex}$
            Send $V'_G$ to $Successor(s)$
```

Fig. 3 Processing of V'_G at p_s in step s of round r

5.1 Content vectors

Our procedures require that each provider p_s summarize terms within its shared content through a bit vector V_s called its *content vector*. For example, a content vector might be a bloom filter [6] of system-specified length L which is formed as follows. Each provider p_s initializes its V_s by setting each bit to 0. Next, for each term t appearing in its shared content, the provider uses a specified hash function H with range $1, 2, \ldots, L$ to set position $H(t)$ in V_s to 1.

The content vectors thus formed are summaries of shared content at a provider. For example, a bloom filter V_s can be used to deduce if a provider p_s is sharing a document with term t as follows. We can hash t to a bit position in the bloom filter from provider p_s. If the bit is 0, then it is guaranteed that p_s shares no documents containing t. If the bit is 1, then the term might or might not occur at p_s since multiple terms might hash to the same value thus setting the same bit in V_s. The probability that such conflicts occur can be reduced by increasing the length L and/or using multiple hash functions. For the sake of our example, assume that $L = 10$. Thus, each provider will have a content vector of length 10.

5.2 Privacy groups

The construction process starts by partitioning the space of providers into disjoint *privacy groups* of size $c > 2$ each. As we show later, the size of a privacy group is proportional to the degree of privacy enjoyed by each participant. Assume for now the partitioning scheme assigns members to groups at random.

For each privacy group G, the providers are arranged in a ring $p_1, p_2, \ldots, p_c$ (see Fig. 2). We use the terms *successor* and *predecessor* of a provider in the usual way with respect to this ordering, with the additional requirement of p_1 being defined as the successor of p_c (and p_c the predecessor of p_1). For our example, assuming that $c = 4$, we must group the providers into two rings of four providers each. For now, we assume that the providers are simply grouped in sequence. Thus, the providers $\{P_1, P_2, P_3, P_4\}$ form the first group, while the providers $\{P_5, P_6, P_7, P_8\}$ form the second group. Clearly, when assigning at random, the arrangement could be quite different. However this simple arrangement makes it easier to explain the remaining steps.

5.3 Group content vectors

Define the *group content vector* of a group G as the vector V_G resulting from performing a logical OR of the set of all content vectors from each provider in G. The next part of the construction is a randomized algorithm for generating the group content vector.

The construction involves performing r rounds in which a vector V'_G is passed from provider to provider along the ring. Each provider, upon receiving the vector, performs the bit-flipping operations outlined in Fig. 3 before passing the vector on to its successor. After r trips around the ring, the vector is sent to a designated index host.

The vector V'_G is initialized by p_1 to a vector of length L with each bit independently set to 0 or 1 with probability 1/2. Each round is associated with probabilities P_{in} and P_{ex} such that $P_{\text{in}} + P_{\text{ex}} = 1$ with $P_{\text{ex}} = 1/2$ initially. After each round, P_{ex} is halved and P_{in} set appropriately.

This process of randomly flipping bits in V'_G is designed so that the end result tends towards the group content vector with high probability as we show (Lemma 5.2). Randomization of the bit flips is used to prevent a malicious provider within the provider group from being able to determine with any certainty the value of bits in the content vector of other providers (Theorem 5.6).

We now go through the process with our example problem. Since the same operations occur for each group, we only go through the process for one group. Table 1 shows the content vectors of the providers in the first group along with their group content vector (logical OR) that must be constructed

Table 1 Content vectors for the providers and the required final Logical OR

V1	V2	V3	V4	Logical OR
0	0	1	1	1
1	0	1	0	1
0	0	1	0	1
0	1	0	0	1
0	0	0	0	0
0	0	0	0	0
0	0	1	0	1
1	0	0	0	1
0	0	0	0	0
0	0	0	0	0

securely. Table 2 presents the iterations of the index construction algorithm, showing the intermediate result in each round, along with its evolution as it goes through each of the providers. Assuming $r = 3$, the algorithm goes through three rounds. The corresponding probabilities P_{ex}, and P_{in} are also displayed, along with the difference between the intermediate result and the real group content vector at the end of each round. One can now infer the working of the algorithm. For example, in round 1, the random initial bit at position 1 is 1 while $V_1[1] = 0$. Therefore with probability $P_{\text{ex}} = 0.5$, this bit is flipped. In this particular case, the bit does not get flipped, when it goes to provider $P2$. However, in bit position 10 the input is 1 while $V_1[10] = 0$, and it does get flipped. Similar changes occur throughout the rounds at different providers. It can be easily seen that just 3 rounds are sufficient for the computed result to exactly match the real group content vector. Running through additional rounds will keep matching this with very high probability.

5.4 Global index

After the r bit-flipping rounds are complete, the vector V'_G from each provider group is sent to a designated index host. The index host receives these vectors from each privacy group along with a list of all providers in the privacy group. It then aggregates these vectors into a materialized index MI. The MI maps a bit position i to a list of providers that belong to privacy groups whose content vector has i set to 1. More formally, $MI(i) = \{p | p \in G \wedge V'_G[i] = 1$ *for some privacy group* $G\}$. With respect to our example the group vectors created for the two groups would be aggregated to create the final materialized index.

5.5 Querying with PPI

Recall that a PPI must map each query q to a set M_q that corresponds to one of the three cases defined in Sect. 4. So far we have defined our MI mapping to map *bits* to providers, but the process of using MI as a PPI that maps *queries* to providers is straightforward: M_q is formed by first taking the conjoined terms Q specified in q and looking up each term's bit position $1 \ldots L$ in MI using the system-specified lookup (hash) function H. The provider list is formed by taking the intersection of $MI(i)$ for each such bit. More formally, $M_q = \cap_{t \in Q} MI(H(t))$. The MI thus serves as an implementation of PPI.

5.6 Correctness

We now show that the mapping PPI from a query q to provider set M_q is expected to satisfy the conditions required of a PPI. First, we show that the set of providers M_q contains all providers that share documents matching q, which is a necessary condition for cases [A] and [B] of the definition.

Table 2 Algorithm iterations

Init	Round 1 $P_{\text{ex}} = 0.5$, $P_{\text{in}} = 0.5$				Round 2 $P_{\text{ex}} = 0.25$, $P_{\text{in}} = 0.75$				Round 3 $P_{\text{ex}} = 0.125$, $P_{\text{in}} = 0.875$			
	P1	P2	P3	P4	P1	P2	P3	P4	P1	P2	P3	P4
1	1	1	1	1	0	0	0	1	1	1	1	1
1	1	0	1	0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	1	1	1	1	1	1
0	0	1	1	0	0	1	1	0	0	1	1	1
1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0
	Difference = 6				Difference =3				Difference = 0			

Lemma 5.1 *Assuming each vector V'_G used to create the mapping PPI is equivalent to or subsumes the group content vector V_G of its group G, the mapping M_q contains all providers that share a document matching q.*

The above claim follows from the simple fact that group content vectors are a logical OR of the individual content vectors from each group member. Thus, each list obtained from PPI given some term t is guaranteed to contain all providers sharing documents with the specific term. Now we establish the qualifying assumption.

Lemma 5.2 *Let c be the number of providers in a privacy group G. For any $0 < \epsilon < 1$, at the end of $r \geq max(3, -\log[1 - \{\frac{8}{7}(1-\epsilon)\}^{1/(c-1)}])$ rounds, a bit b that is 0 in V_G is also 0 in V'_G with probability $1 - e^{-c}$, while a bit b that is set to 1 in V_G is also 1 in V'_G with probability $1 - \epsilon$.*

Proof Let us first consider the case when $b = 0$ in V_G. This means that none of $p_1, p_2, \ldots, p_c$ will set b to 1. If b was 0 in V'_G at the start of the construction, it stays 0 until the end. If b was 1 at the start of the construction, each provider in G will attempt to reset it at each step of every round with probability p_{ex} of the round. The probability that b is still 1 at the end of r rounds is $\Pi_{i=1}^{r}(1 - 1/2^i)^c \leq e^{-c}$.

Now consider the case when $b = 1$ in V_G. Note that in the worst case, only p_1 has $b = 1$ in V_1 and the rest of p_i attempt to set b to 0. Consider the value of bit b at the start of the rth round. Let b be 0 with probability P_0 and 1 with probability P_1. In the rth round, $P_{\text{ex}} = 1/2^r$ and $P_{\text{in}} = 1 - P_{\text{ex}}$. The bit b is 1 at the end of round r with probability $P(b = 1) = (1 - P_{\text{ex}})^{c-1}(P_1 + P_{\text{in}} P_0) = (1 - P_{\text{ex}})^{c-1}(P_1 + (1 - 1/2^r)P_0) = (1 - P_{\text{ex}})^{c-1}(P_1 + P_0 - P_0/2^r)$. But $P_1 + P_0 = 1$ and $P_0 \leq 1$. This means $P(b = 1) \geq (1 - P_{\text{ex}})^{c-1}(1 - 1/2^r)$. For $r \geq 3$, $(1 - 1/2^r) \geq 7/8$. For any $0 < \epsilon < 1$, we can ensure that $b = 1$ with probability $1 - \epsilon$ by requiring that $P(b = 1) \geq (1 - P_{\text{ex}})^{c-1}\frac{7}{8} \geq 1 - \epsilon$ or $(1 - 1/2^r) \geq [\frac{8}{7}(1-\epsilon)]^{1/(c-1)}$ or $r \geq -\log[1 - \{\frac{8}{7}(1-\epsilon)\}^{1/(c-1)}]$. □

Lemma 5.2 shows that we can make V'_G subsume the group content vector V_G with probability arbitrarily close to one by increasing the number of rounds. Henceforth, we assume that the number of rounds has been appropriately chosen so that we can safely assume subsumption. Given this assumption, we have established the following:

Theorem 5.3 *For any query q with conjoined terms Q, $M_q = \cap_{t \in Q} PPI(H(t))$ contains all providers that share documents matching q.*

All that remains to establish that the mapping M_q is expected to meet the requirements of a PPI is to demonstrate that should M_q be non-empty, then it is expected to contain at least half false positives, or be equivalent to the entire provider set.

Lemma 5.4 *Let n be the number of providers indexed by PPI. For query q with selectivity σ, the expected number of groups that have a provider sharing a document matching q is $\frac{n}{c} \times [1 - (1-\sigma)^c]$.*

Proof The probability that *no* provider in a group shares a document matching a query with selectivity σ is $1 - \sigma$ multiplied across all group members, or $(1-\sigma)^c$. The expected number of groups that share at least one such document is thus one minus this number multiplied by the total number of groups, or $\frac{n}{c} \times [1 - (1-\sigma)^c]$. □

Case [B] asks that a query q with selectivity $\sigma < 0.5$ be mapped to at least $2\sigma n$ providers. The construction ensures that should one provider in a privacy group share a document matching a query, that all other members of its group will be contained in the mapping M_q. From Lemma 5.4 we know that a query q with selectivity σ is mapped to $n[1 - (1-\sigma)^c]$ providers. Thus, Case [B] holds if $2\sigma n \leq n[1 - (1-\sigma)^c]$ or $2\sigma + (1-\sigma)^c \leq 1$. As c increases, values of σ that satisfy the equation will increase. For $c = 4$, the condition is expected to hold for $0 < \sigma \leq 0.456$, while for $c = 10$, the values are $0 < \sigma \leq 0.499$.

What if σ for a query q lies beyond this range? The definition states that the constructed index must follow (the only remaining) Case [C]. In other words, the term must map to n providers. For the constructed index, this is not true as $n[1 - (1-\sigma)^c] < n$ for all c and $\sigma \neq 1$. However, we can make $n[1 - (1-\sigma)^c] \rightarrow n$ by increasing c. For example, for $c = 4$, $n[1 - (1-\sigma)^c] \geq 0.937n$ for $0.5 \leq \sigma \leq 1$ while $c = 10$ leads to a value of $0.999n$ for the same range of σ.

Theorem 5.5 *The mapping M_q of providers produced by PPI for any query q is expected to satisfy the conditions for being privacy-preserving.*

Note that we have not proved that the mapping M_q is *guaranteed* to be privacy-preserving. As we have quantified above, there is a small chance that for any particular query, there may not be sufficient false positives, or that the result will not quite contain the list of all providers when query selectivity is low. Nevertheless, the data points we presented above show that the probability is quite low for reasonable settings of c. With respect to any given query, then, the output of our index is *expected* to be privacy-preserving, and in fact this expectation is near one for reasonable settings of c. Also note there is no telling *which* of the queries lead to output that does not precisely satisfy the definition, which limits the opportunities for an adversary to actually exploit this fact.

We wish to emphasize that these exceptional cases are only with respect to having a sufficient number of false positives. So long as accurate group content vectors are constructed (which can be guaranteed with high probability), Theorem 5.3 implies correctness of the search output.

5.7 Privacy characteristics

Let us next consider the privacy breaches that could occur during index construction. Note that the final group content vectors V'_G do not provide an adversary with information that cannot already be obtained from directly inspecting the PPI itself. This gives an adversary who is outside the provider group of another provider p_s no more information about p_s than what is available to the adversarial searcher discussed previously in Sect. 4.

What remains to be quantified then are the privacy breaches that occur between members of a group G while V'_G is being generated. The communication between group members consists of sending the current working vector V'_G from one provider to its successor. We assume all communications are encrypted and authenticated, and thereby immune to interception by anyone other than the intended recipient. Each member in G thus has a limited view of the construction process. Still, the following theorem shows that this limited view does leak some information that can be used for privacy breaches by an active adversary within the group.

Theorem 5.6 *Within a privacy group G, an active adversary p_a can learn that its predecessor p_s is sharing a document containing term t with probability up to* 0.71.

Proof The content vector V'_G that p_a receives could have been altered by any of the remaining $c-1$ $(c > 2)$ providers but was modified by p_s most recently. The adversary p_a can remember the content vectors from each round and attempt to deduce bits being set by p_s. Trivially, p_a can deduce that none of the members have terms that result in bits $b = 0$ in V'_G at the end of r rounds. However, for p_s it can deduce more. If p_a observes a bit $b = 0$ in V'_G at the end of a round, and that bit is subsequently 1 for all remaining rounds r', then it can deduce that p_s does not have a term that sets bit $b = 0$ with probability $1 - P_{\text{in}} = P_{\text{ex}} = 1/2^{r'}$. Moreover, if it observes that a bit $b = 1$ in all the r rounds, then the probability that p_s does not have the term is $\Pi_{i=1}^{r}(1 - 1/2^i)$ which tends to the limit of 0.29 from above. In other words, p_a has deduced that p_s has a document containing the term with probability up to 0.71. □

More problematic is the ability with which colluding adversaries may breach privacy. In the worst case, colluding adversaries may be serving as both the predecessor and successor of a provider p_s. In such a case, the adversaries can determine precisely the bits flipped by p_s during each round. The adversaries can then determine the content vector of p_s with high probability (a privacy breach tending towards Provable Exposure).

We suggest that such attacks can be made irrelevant if a provider can ensure that neighboring providers in its group are "trustworthy". This can be achieved in practice by having providers arrange themselves along rings according to real-world trust relationships. However, in many situations this may be very difficult or even impossible—in such cases technical solutions are necessary. In Sect. 6, we present a technical solution that solves this problem—a deterministic secure procedure for generating a privacy-preserving index using cryptographic techniques. However, the downside is that it is less efficient than this protocol.

5.8 Efficiency

An ideal PPI is, by definition, at most 2× less selective than a precise index for any given query. The index constructed by our algorithm maps queries to groups of size c. In the worst case from the perspective of efficiency (but best case from the perspective of privacy), for a given query q, only one member of each identified group actually shares a matching document. In such a case the index is $c\times$ less selective than a precise index. The worst-case loss in efficiency of the constructed index is thus proportional to the degree of privacy (c) desired by its participants. However, the expected number of providers for a query q of selectivity σ is $n[1 - (1 - \sigma)^c]$ by Theorem 5.4. The loss in selectivity is then $[1 - (1 - \sigma)^c]/\sigma$. Figure 4 plots the loss in selectivity for different selectivity levels ranging from 0.001 to 1.0 with increments of 0.001, for different ring sizes. It can be clearly seen, that the worst case is approached only for very low selectivity levels. The drop off for each curve is very quick. Indeed, assuming that queries are uniformly distributed (w.r.t the selectivity), the average loss in selectivity is 2.9245 (as computed from the figure). Given that, by definition, a privacy-preserving index must be atleast two times as less selective compared to the original index, this is quite good. Even if queries are not uniformly distributed, the quick drop off leads to reasonably good results. For example, with $\sigma = 0.1$, the loss is 3.43 for $c = 4$ and 6.51 for $c = 10$.

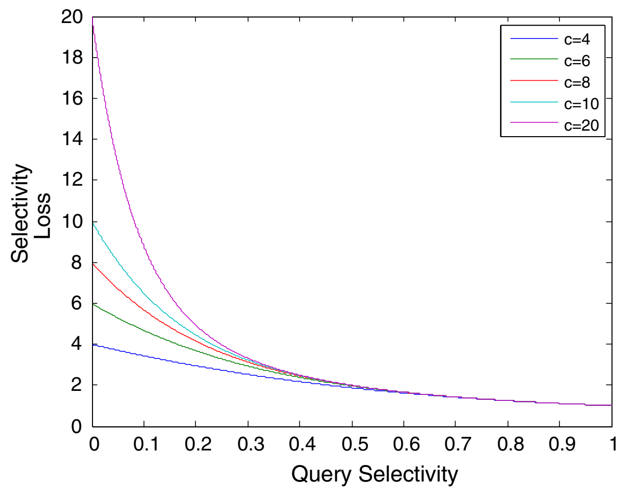

Fig. 4 Selectivity loss behavior

We discuss such expected loss in selectivity in more detail in Sect. 7.

Theorem 5.7 *A query q of selectivity σ has a worst-case selectivity of $c \times \sigma$ and an expected selectivity of* $[1-(1-\sigma)^c]$ *in the constructed index.*

6 Constructing a PPI securely

The randomized procedure for constructing a PPI as discussed above is unable to provide the stronger privacy guarantees of Probable Innocence in two cases: (a) Providers who immediately precede an active adversary are assured only Possible Innocence, and (b) Providers who have both their neighbors colluding against them can be Provably Exposed. We now present a deterministic procedure for generating a group content vector that eliminates the two privacy breaches. The procedure uses a probabilistic homomorphic encryption system which is described in general terms. The procedure pays a price in efficiency for its stronger privacy guarantees as we discuss later. As before, all of the providers are arranged in a ring. However, only a single round is needed, instead of the multiple rounds required in the randomized procedure. The basic idea is for the initiator to suitably encrypt its bit vector and pass it along to the next party. Each successive party suitably modifies the encrypted vector (operating on it in while encrypted), until it can finally be decrypted to give only the global index. To explain this further, we first describe the underlying encryption scheme and then give the actual algorithm.

6.1 Building block: generic encryption system

We present our algorithm in a generic setting using the notation of Stern [36] to describe the homomorphic probabilistic encryption system required. The required system consists of:

A *Security parameter* k: This is the standard security parameter that is fixed at the time the cryptosystem is set up. Any adversary's computing power (algorithm running time) is assumed to be bounded by a polynomial function in k (thus, to protect against stronger adversaries, one only needs to increase the value of k). k also determines the size of the plain text messages and cipher text. Thus, $U(k)$, $X(k)$ and $Y(k)$ are defined as $U(k) : \{a | a \in \{0,1\}^{u(k)}\}$. Similarly, $X(k) : \{a | a \in \{0,1\}^{x(k)}\}$, and similarly for $Y(k)$. Here, $u(k)$, $x(k)$, $y(k)$ are polynomial functions in k used to fix an appropriately sized message space (indeed, they possess different values in each actual encryption system used in practice).

B *Probabilistic functions* f and g: used for encryption $f : U(k) \times X(k) \to Y(k)$, and decryption $g : Y(k) \to X(k)$ respectively such that $(\forall (u,x) \in U(k) \times X(k))$ $g(f(u,x)) = x$. The encryption function is public, while the decryption function is private. Note that the existence of a decryption algorithm implies that the function is injective with respect to its second parameter, that is, for $(u_1, x_1), (u_2, x_2) \in U(k) \times X(k)$, if $f(u_1, x_1) = f(u_2, x_2)$, then $x_1 = x_2$

We impose three additional properties on f and g:

B.1 The encryption function f should be homomorphic, that is: $\forall (u_1, x_1), (u_2, x_2) \in U(k) \times X(k)$, $f(u_1, x_1)$ $f(u_2, x_2) = f(u_3, x_1 + x_2 \bmod x(k))$ where u_3 can be computed in polynomial time from u_1, u_2, x_1 and x_2.

B.2 The encryption function f should be *semantically secure* [25]. Informally, this means that for a polynomially bounded adversary, the analysis of a set S of ciphertexts does not give more information about the cleartexts than what would be available without knowing S.

B.3 There exists a "hiding" function $hide : U(k) \times Y(k) \to Y(k)$, depending only on the public parameters of the system such that: $\forall (w, x) \in U(k) \times X(k), \forall u \in U(k)$, $hide(u, f(w, x)) = f(uw' \bmod w(u), x)$ where w' can be computed in polynomial time from w, x. Indeed, $hide$ can be defined by $hide(u, x) = f(u, 0) * x$.

Several encryption systems (e.g., Goldwasser-Micali [7], Benaloh [5], Naccache-Stern [30], Okamoto-Uchiyama [32]) satisfy all of the three properties required above. Any of these cryptosystems can be used in our algorithm described next (they each simply have different values for $x(k)$, $y(k)$ and $u(k)$.

6.2 Secure group content vectors

As before, the procedure starts with each provider summarizing terms within its shared content through a bit-vector V_s. The space of providers is then partitioned into disjoint privacy groups of size $c > 2$ each. Each group fixes one of the suitable cryptosystems described above. The construction procedure itself is different from the previous discussion, requiring only a single round along the ring to construct the group content vector.

The first provider, p_1, generates a public key E and private key D for the chosen cryptosystem. The public key, E, is sent to all the parties. The private key, D, is secret and known only to p_1. The procedure constructs an encrypted group content vector $EncV_G$ which is decrypted by p_1 using D to obtain V_G. The group content vector V_G is sent to the designated index host. The index host receives these vectors from each

Fig. 5 Processing of $EncV_G$ at p_s in step s

```
SECURECONSTRUCTION($V_s$, $EncV_G$)
  for ($i := 1; i <= L; i++$)
    do
      if ($V_s[i] = 0$)
        then $EncV_G[i] = hide(w, EncV_G[i])$, ( $w$ RANDOMLY CHOSEN FROM $U(k)$ )
        else $EncV_G[i] = E(w, V_s[i])$, ( $w$ RANDOMLY CHOSEN FROM $U(k)$ )
```

privacy group along with a list of all providers in the privacy group. It then aggregates these vectors into a materialized index MI and enables queries on the MI as before.

The vector $EncV_G$ is initialized by p_1 to a vector of length L with each bit independently set to an encryption of O (i.e., $\forall i$, $EncV_G[i] = E(w, 0)$, with w chosen uniformly at random from $U(k)$). The vector $EncV_G$ is passed along the ring. Each provider upon receiving $EncV_G$, performs the actions outlined in Fig. 5 before passing the vector on to its successor.

The steps are designed to allow a provider p to set the ith element of $EncV_G$ to a random encryption of 1 if the ith bit of V_p is 1. Otherwise, the ith element of $EncV_G$ is kept the same, but obfuscated by setting it to a different encryption of the same clear-text. Finally, once the round is over, p_1 decrypts the vector it gets with the private key D to retrieve the group content vector V'_G, i.e., $V'_G = D(EncV_G)$. Note that the protocol works even if p_1 is a third party that simply collects the group content vector. It could also be one of the providers in another group.

With respect to our expository example, we go through the working of the algorithm, just as in Sect. 5 with the same provider content vectors, and parameter values. As earlier, we simply go through the index construction process for the group consisting of providers $\{P1, P2, P3, P4\}$. Table 3 shows the single round that the secure construction procedure goes through. As discussed above, the initial vector is set to random encryptions of 0. All of the r_i stand for random values selected by each provider as necessary. The r_i* (correspondingly r_i**, and r_i***, represent the effect of the hide operation when carried out over the input r_i (correspondingly r_i*, and r_i**). For example, on receiving the first bit, since $V_1[1] = 0$, it merely hides what it receives, thus sending out $E(r_1*, 0)$. On the other hand, for bit position 2, $P3$ receives $E(r_{13}*, 1)$, but since $V_3[2] = 1$, it send out a fresh encryption of 1, namely, $E(r_{14}, 1)$. At the end, the final vector can be decrypted to receive the real group content vector. The rest of the steps after this are as earlier.

6.3 Correctness

We now show that the mapping MI constructed using the cryptographic procedure satisfies the conditions required of a

Table 3 Secure construction procedure example

Init	P1	P2	P3	P4
$E(r_1, 0)$	$E(r_1*, 0)$	$E(r_1**, 0)$	$E(r_{11}, 1)$	$E(r_{12}, 1)$
$E(r_2, 0)$	$E(r_{13}, 1)$	$E(r_{13}*, 1)$	$E(r_{14}, 1)$	$E(r_{14}*, 1)$
$E(r_3, 0)$	$E(r_3*, 0)$	$E(r_3**, 0)$	$E(r_{15}, 1)$	$E(r_{15}*, 1)$
$E(r_4, 0)$	$E(r_4*, 0)$	$E(r_{16}, 1)$	$E(r_{16}*, 1)$	$E(r_{16}**, 1)$
$E(r_5, 0)$	$E(r_5*, 0)$	$E(r_5**, 0)$	$E(r_5***, 0)$	$E(r_5****, 0)$
$E(r_6, 0)$	$E(r_6*, 0)$	$E(r_6**, 0)$	$E(r_6***, 0)$	$E(r_6****, 0)$
$E(r_7, 0)$	$E(r_7*, 0)$	$E(r_7**, 0)$	$E(r_{17}, 1)$	$E(r_{17}*, 1)$
$E(r_8, 0)$	$E(r_{18}, 1)$	$E(r_{18}*, 0)$	$E(r_{18**}, 1)$	$E(r_{18}***, 1)$
$E(r_9, 0)$	$E(r_9*, 0)$	$E(r_9**, 0)$	$E(r_9***, 0)$	$E(r_9****, 0)$
$E(r_{10}, 0)$	$E(r_{10}*, 0)$	$E(r_{10}**, 0)$	$E(r_{10}***, 0)$	$E(r_{10}****, 0)$

PPI. We first show that correctness is achieved as the assumption in Lemma 5.1 holds true.

Lemma 6.1 *Let c be the number of providers in a privacy group G. At the end of the group content vector construction procedure, a bit b that is* 0 (*resp.* 1) *in V_G is also* 0 (*resp.* 1) *in V'_G.*

Proof All of the bits in the vector under go the same operations. Thus to prove the correctness of the entire construction procedure, it is sufficient to prove that a single bit of $\vec{V'_G}$ is the exact logical OR of the corresponding bits of the vectors $\vec{V_1}, \vec{V_2}, \ldots, \vec{V_c}$ (i.e., we just need to prove that $V_G[i] = V_1[i] \vee V_2[i] \cdots \vee V_c[i]$).

Originally the vector $EncV_G$ is set to random encryptions of 0. If $V_j[i] = 1$, then p_j sets $EncV_G[i]$ to a random encryption of 1. Otherwise, $EncV_G[i]$ is simply set to a new encryption of the same element. Thus, $EncV_G[i] = E(u, 0)$ if and only if $\forall j, V_j[i] = 0$. Otherwise $EncV_G[i] = E(u, 1)$, for some $u \in U(k)$. Thus the decryption of $EncV_G$ ($V'_G = D(EncV_G)$) correctly gives the logical OR of the corresponding bit in the vectors $V_1, V_2, \ldots, V_c$. $\square$

With Lemma 5.1 established, it is easy to see that Theorems 5.3 and 5.5 continue to hold for the new construction procedure, establishing the correctness of the constructed index.

6.4 Privacy

We now establish the privacy of our procedure. First, we prove that our protocol is secure assuming *semi-honest* adversarial behavior. We then discuss how resilient the protocol is to collusion and present a way to make the protocol completely resistant to collusion.

Informally, a semi-honest provider follows the rules of the protocol using its correct input, but is free to later use its records from execution of the protocol to enable privacy breaches of other providers in the group. Formally, the behavior can be modeled as follows:

Definition 1 (privacy w.r.t. semi-honest behavior) [23]:

Let $f : \{0,1\}^* \times \{0,1\}^* \longmapsto \{0,1\}^* \times \{0,1\}^*$ be a probabilistic, polynomial-time functionality, where $f_1(x, y)$(resp., $f_2(x, y)$ denotes the first (resp., second) element of $f(x, y)$). Let Π be a two-provider protocol for computing f.

The *view* of the first (resp. second) provider during an execution of Π on (x, y), denoted $VIEW_1^{\Pi}(x, y)$ (resp., $VIEW_2^{\Pi}(x, y)$) is $(x, u, m_1, \ldots, m_t)$ (resp., $(y, u, m_1, \ldots, m_t)$). u represent the outcome of the first (resp., second) provider's internal coin tosses, and m_i represents the ith message it has received.

The *output* of the first (resp., second) provider during an execution of Π on (x, y), denoted $OUTPUT_1^{\Pi}(x, y)$ (resp., $OUTPUT_2^{\Pi}(x, y)$), is implicit in the provider's view of the execution.

Π privately computes f if there exist probabilistic polynomial time algorithms S_1 and S_2 s.t.

$$\{(S_1(x, f_1(x, y)), f_2(x, y))\}_{x,y\in\{0,1\}^*}$$
$$\equiv^C \left\{\left(VIEW_1^{\Pi}(x, y), OUTPUT_2^{\Pi}(x, y)\right)\right\}_{x,y\in\{0,1\}^*}$$
$$\{(f_1(x, y), S_2(x, f_1(x, y)))\}_{x,y\in\{0,1\}^*}$$
$$S \equiv^C \left\{\left(OUTPUT_1^{\Pi}(x, y), VIEW_2^{\Pi}(x, y)\right)\right\}_{x,y\in\{0,1\}^*}$$

where $\equiv^C$ denotes computational indistinguishability.

Goldreich [23] shows that in such a model, computing a function privately is equivalent to computing it securely.

Privacy by simulation The above definition says that a computation is secure if the view of each provider during the execution of the protocol can be effectively simulated given the input and the output of that provider. Thus, in the proof of security, we only need to show the existence of a simulator for each provider that satisfies the above equations. The way we prove this is by showing that we can simulate each message received. Once the received messages are simulated, the algorithm itself can be used to simulate the rest of the view. Note that the simulator doesn't generate exactly the same message, but the *probability* that the simulator produces a given message is the same as the *probability* that message is seen in a run of the real algorithm, regardless of the input data. The (random) choice of encryption keys ensures that even with fixed input data, different runs of the algorithm will produce different messages—but if the distribution of the messages can be simulated, the view is equivalent to the simulation and Definition 1 is satisfied.

This does not quite guarantee that private information is protected. Whatever information can be deduced from the final result is not kept private. For example, if the logical OR vector is composed entirely of 1s then knowing its own vector, a provider can figure out that at least one of the other parties supports/has a 1 in a bit where it has a 0. Here, the result reveals information to a provider. The key to this definition is that nothing *beyond* the results is learned.

In summary, a secure protocol will not reveal more information to a particular provider than the information that can be induced by looking at that provider's input and the final output.

Theorem 6.2 *The cryptographic algorithm computes V'_G without revealing anything to any provider other than its input. Only p_1 learns the final output, V'_G.*

Proof A simulator is presented for the view of each provider. We only show how to simulate the messages received. The rest of the proof trivially follows from this.

Let us first consider the case of providers $p_2, p_3, \ldots, p_c$. Every provider receives the public key E from p_1. This can be simulated simply by randomly choosing a key E over the space of possible keys. Next, the vector $EncV_G$ needs to be simulated. Every bit of $EncV_G$ is simulated by randomly choosing a bit b(0 or 1) and uniformly choosing a random w from $U(k)$, and computing $E(w, b)$. The semantic security property of the encryption system guarantees that no advantage or information can be gained from the cipher-text resulting from the encryption algorithm (even while knowing the public key, as long as the private key is secret). In other words, it is not computationally possible to distinguish between the encryption of a 0 or a 1 when w is randomly chosen with uniform probability over $U(k)$. Thus, by selecting random values for w and b, the encrypted message generated is computationally indistinguishable from the message received.

Now consider the case of provider p_1 which actually generates a random encryption key E, as well as the original vector V'_G (random encryptions of 0). The only message received is the final encrypted vector $EncV_G$. This cannot be generated at random since the decryption of the vector has to match the final result. However, since the final result V'_G is known to p_1, for each bit $V'_G[i]$, p_1 simply generates a random encryption of that bit (choose a random w from $U(k)$ and compute $EncV_G[i] = E(w, V'_G[i])$. Thus p_1 can generate the entire vector $EncV_G$. This is computationally indistinguishable from the message it receives since in both cases (P_c as well as the simulator)

- w is chosen randomly with uniform probability from $U(k)$,
- generate a random encryption of the same actual cleartext bit $V'_G[i]$. □

The semi-honest model is not actually a limiting factor for our protocol. Though we have proved the security of our protocol under the semi-honest model, it is actually resilient to more powerful adversaries. However, one assumption that we make is that secure authenticated communications are used—this is quite easy to ensure, using PKI for example, and is outside the scope of this paper. With this assumption, it is easy to show the resiliency of our protocol. Since only a single round of communication takes place, either a participant can refuse to interact or else simply manipulate the message it sends forward. Refusal to interact can be directly detected and dealt with, outside of the protocol. Manipulation of a message can only lead to change in the created index (however this can also be easily done through manipulation of input, which is quite outside the scope of even the malicious model). When threshold encryption is used to make the protocol collusion resistant (this is discussed in detail below), there is no way in which any provider can learn the private input of any other provider. cannot learn anything more about any other provider's vector. Thus, in this sense, our protocol is resilient to malicious adversaries. This makes the protocol quite secure.

6.4.1 Collusion

We now discuss how resilient the protocol is to collusion. In fact, collusion affects the protocol much less. The predecessor and successor of a provider cannot collude against the provider to reveal any additional information. Since the encryption is semantically secure, without knowing the decryption key, either 0 or 1 is equally likely. The only possibility for collusion is if p_1 (knowing D) colludes with any other provider p_i. In this case, the OR vector upto p_{i-1} will be revealed to p_i. However, based on trust relations, an appropriate provider can be chosen to do key generation. The trust requirements here are much reduced from those required by the randomized procedure of the previous section.

With a minor modification, we can easily create a completely secure protocol, that eliminates even the trust requirements on p_1 and hence prevents collusion breaches. The key idea is to use a stronger encryption system, specifically the *threshold homomorphic encryption* system, during group content vector construction. A *threshold homomorphic encryption* system is an encryption system that has all the properties of a standard homomorphic encryption system (presented above) and has the following additional properties:

C the public key E is known to all providers,
D the decryption key D is shared between all the providers,
E any provider can encrypt a message, and
F decryption of a message requires participation by at least T providers.

By setting the threshold $T = c$, we can require the participation of all providers for any decryption step. A formal definition of threshold homomorphic encryption systems can be found in [11]. A number of such systems do exist [16,12]. When such a completely secure threshold encryption scheme is used, the protocol is actually completely resistant to collusion, even in the malicious model. Due to the threshold encryption no provider can decrypt any of the messages without the help of all of the other parties. As long as at least one party remains true, even if all of the others collude against it, its security is guaranteed. Now, collusion can no longer occur, and the system is perfectly secure. The only downside with using threshold encryption is that of efficiency, which we discuss in the following section. Thus, the cryptographic protocol is completely secure, and can be used to construct the privacy-preserving index, without any leakage of information, or any trust requirements.

7 Empirical evaluation

In this section, we evaluate the behavior of analytical bounds established in previous sections, and also the performance of the indexing scheme on real data. We analyze both algorithms (randomized and cryptographic) as well as the overall search methodology. Specifically, we show that:

A The number of rounds required by the randomized index construction algorithm is small in practice, leading to efficient index creation.
B The probability with which an adversary provider can breach privacy of its predecessor during the randomized index construction algorithm tends quickly to our bound of 0.71, which implies that careful tuning of the other problem parameters is unlikely to be useful for avoiding this problem in practice.
C On real data, reasonable settings of ϵ ensure the generated index during the randomized index construction suffers no loss of recall.
D The computation and communication costs scale linearly with the size of the group for the cryptographic index construction procedure.

E The randomized procedure is computationally significantly more efficient than the cryptographic method for large datasets.

F On real data, the performance penalty of a PPI compared to a precise provider index is roughly $\frac{2}{3} \times c$ when averaged over all queries.

7.1 Choice of rounds for randomized construction

We start our evaluation of the index by studying the number of rounds required for its construction in a privacy group using the randomized procedure. As discussed in Theorem 5.2, the number of rounds w required in a group G of size c for ensuring $V_G \subseteq V'_G$ with probability $1-\epsilon$ for $0 < \epsilon < 1$ is given by $w \geq max(3, -\log[1-\{\frac{8}{7}(1-\epsilon)^{1/(c-1)}\}])$. Thus, w depends on c and ϵ. We note that w is proportional to the processing and bandwidth costs incurred during index construction.

Figure 6 plots the value of w for various c and ϵ. The X-axis plots c on a logarithmic scale while the Y-axis plots w. The number of rounds w grows almost linearly with logarithm of the size c of a privacy group. Since c determines the level of privacy ensured for a member of G, index construction scales well with the privacy requirements imposed on it.

The curves in Fig. 6 pull up parallel to each other for increasing ϵ values. As ϵ increases, accuracy of the group vector increases. As can be observed, an increase in accuracy $10\times$ (from $\epsilon = 0.100$ to $\epsilon = 0.01$ and then from $\epsilon = 0.01$ to $\epsilon = 0.001$) results in an average increase in w by a constant value of 2.5 rounds. Thus, the index construction process scales well with desired ϵ.

7.2 Breaches within a privacy group for randomized construction

Theorem 5.6 shows that the privacy breach during randomized construction procedure at a provider is the most severe to the preceding provider in the index construction chain within a privacy group. The privacy breach P_{loss} was quantified as $P_{\text{loss}} = \Pi_{i=1}^{w}(1-1/2^i)$. The function is plotted in Figure 7 for various values of c and ϵ. The X-axis plots the size c of a group G while Y-axis plots P_{loss}.

Figure 7 shows that the privacy breach tends to 0.29 quickly, except for small values of c and ϵ. Still, the absolute difference is quite small. This suggests that careful tuning of c and ϵ cannot be used to avoid this potential privacy breach. However, our suggestion on organizing privacy groups based on real-world trust remains a valid option.

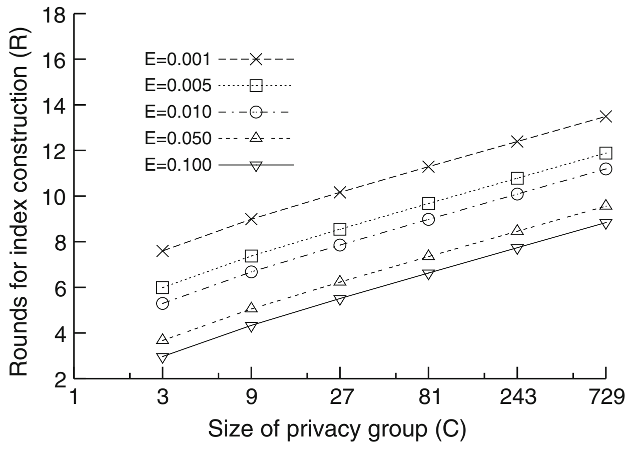

Fig. 6 Number of rounds for index construction

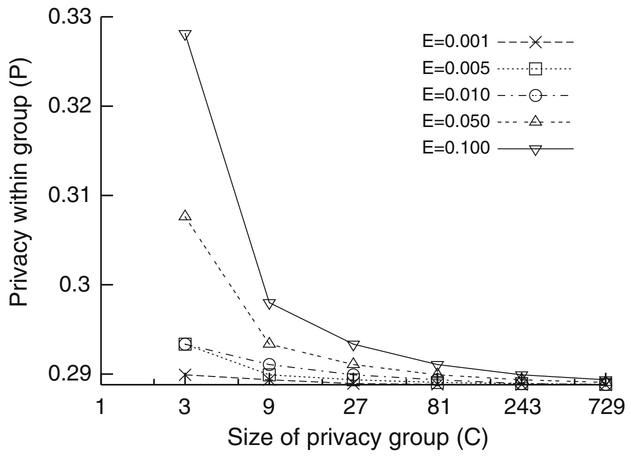

Fig. 7 Loss of privacy during index construction

7.3 Efficiency of cryptographic and randomized procedures

First, we evaluate the communication and computation cost of the homomorphic probabilistic encryption algorithm. The provider p_1 broadcasts the key E to all other parties. Each provider also sends the entire (encrypted) vector to the next provider once. Thus the total communication cost is $(c-1) * keysize + c * L * encrypted\_msg\_size = O(cl)$ bits, and $c-1+c = 2c-1$ messages (assuming the entire vector can be sent off as a single message). Thus, the entire algorithm is quite efficient in terms of communication. In terms of computation, every provider has to perform L encryptions (one for each bit in its vector), and finally p_1 has to perform L decryptions to get the final result. Thus, there is a total of cL encryptions and L decryptions.

We ran tests on a SUN Blade 1000 workstation with a 900 Mhz processor and 1 gigabyte of RAM. A C implementation of the Okamoto–Uchiyama [32] encryption system was used. The key size was fixed at 1152 bits, which is in fact more than necessary for most applications. The computation time

	100	1000	10000	100000
encrypt	1.33s	13.06s	2.2min	21.5min
decrypt	2.11s	20.80s	3.5min	35.3min

Fig. 8 Computation time required for encryption/decryption

required for different values of L are summarized in Fig. 8. It is obvious that the encryption/decryption cost increases approximately linearly with the number of items. Using this table, it is very easy to estimate the actual time required for different number of parties and different vector sizes. For example, five parties with vectors of size 1,00,000 would require approximately 145 minutes. The time required would be significantly lower with smaller key sizes and with use of special purpose encryption hardware. All of this calculation assumes that the entire cost of encryption is borne at run time. However, typically for cryptographic protocols, a lot of the work can be done offline. This is especially true for our protocols. Essentially all of the encryptions can be performed offline, before the protocol starts (since they are all encryptions of 0 or 1). During the protocol, then, the encryption cost is only that of indexing and selecting the appropriate encryption, or that of modular multiplications (due to the invocation of the hide function). Of course, the decryption of the final vector will still have to be done at runtime. The cost of indexing is negligible. Thus for the overall protocol, the entire cost reduces to the cost of modular multiplications, which are quite inexpensive, and that of decryption. We estimated the cost of a modular multiplication using the GMP library on the Sun Workstation, where 1 million multiplications take 55 s. Thus, for the earlier example of five parties with vectors of size 1,00,000, the online computation cost would only be at most about 30 min.

Even without dividing the cost into online/offline costs, the performance of the protocol can be greatly improved with a simple implementation trick. Before starting the actual protocol, each party can generate sufficient number of encryptions. Once this is done by all parties, they can start the actual protocol. Since the encryptions can be done in parallel, the overall cost would only depend on the length of the vector, and is independent of the number of parties. Thus, for the prior example of five parties with vectors of size 1,00,000, the total cost would reduce to only about 50 minutes. Finally, parallelization can significantly reduce this cost, as well. Even a modern 3.0 Ghz Core 2 Duo processor would perform six times as faster, with full utilization. Quad cores, or further parallelization across cores/computes can further significantly reduce this cost. Thus, in actual practice the overall computation cost can be made quite reasonable.

We now consider the cost of the threshold variant of the encryption system. Encryption still takes the same amount of time. Decryption time increases—it is linearly proportional with the number of parties required to do decryption. The

	1000	10000	100000	1000000
3 parties	0.010s	0.048s	0.441s	5.8s
5 parties	0.012s	0.072s	0.709s	8.3s
10 parties	0.018s	0.137s	1.36s	17.45s

Fig. 9 Computation time required for the randomized approach, 30 rounds

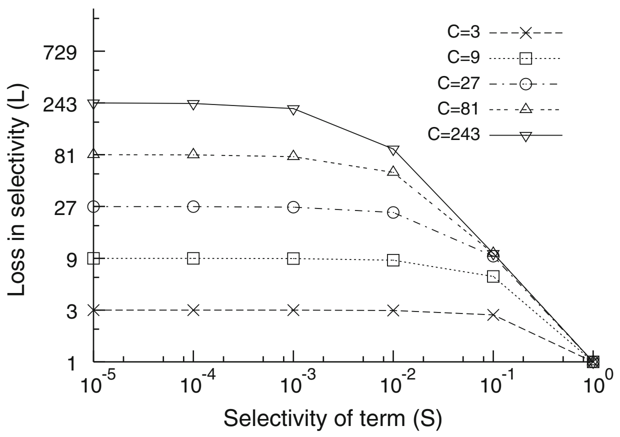

Fig. 10 Loss in selectivity

most inefficient part of the system is key generation (which generates the encryption key and the shares of the decryption key). If we assume this to be done offline, we need not consider it to be a factor in efficiency.

We also measured the computation time of the randomized approach to contrast it with the cryptographic scheme. Figure 9 gives the computation time required for different vector sizes assuming five parties and 30 rounds (which is significantly more than necessary). From this table it is obvious that the computation time of the randomized procedure is negligible. For example, instead of 145 min for the cryptographic approach, the randomized approach would only take 0.7 s for five parties with 1,00,000 items. Indeed, for this approach, the bottleneck would lie in the communication cost instead of the computation cost. It is clear that this approach can easily scale to the order of millions of items. This makes a compelling argument for using it as opposed to the cryptographic approach for really large scale indexes.

7.4 Loss in selectivity in PPI

We next study the expected increase in query processing costs incurred by the use of the constructed PPI. The increase in costs is due to the decrease in selectivity of a term in the constructed index. Theorem 5.7 implied that the expected selectivity in the constructed index for a term t with actual selectivity σ is $[1-(1-\sigma)^c]$. The loss in selectivity can be quantified as the ratio $[1-(1-\sigma)^c]/\sigma$. Figure 10 plots this ratio on the Y-axis against σ on the X-axis for various values of c.

We observe that as $\sigma \rightarrow 1$, the expected loss in selectivity diminishes to 1 for all curves. Indeed, we anticipate such a result as almost all providers share a document with the particular term. Both the precise and the constructed index must now map the term to all providers.

The lower selectivity range is more interesting as it corresponds to rare items that are shared by a few providers. One can argue that it is the rare terms that need to be protected carefully, so that they cannot be pinned to the sharing provider. We observe here that the loss in selectivity for such terms is c. In other words, $c\times$ providers are identified as potentially having a document sharing such a term. The curves taper off when $\sigma nc \geq n$.

7.5 Effectiveness of V'_G in search

Recall that a choice of c and ϵ determines the magnitude of inaccuracy observed in the group content vector. We next study the effects of such inaccuracies on the *search* function. Queries asked by peers in YouSearch against shared content were logged. From a sample of 1,700 logged queries, we randomly selected 100 distinct queries as our query set.

The peers (providers) were randomly partitioned into c sized privacy groups and indexes created for each group for various ϵ values. The hash of each term in a query was computed to determine the corresponding bit in the bloom filters. The determined bit position was checked in the index V'_G for each group G. Groups that had bits for all terms in a query q set to 1 were deemed to be relevant. The query will then be evaluated at each of the member providers. Since group indexes constructed with larger values of epsilon can be expected to have some bits falsely set to 0, the answers for q at such member providers will be lost.

For each query q, define S_q to be the set of providers that have bits for all terms in q set to 1 in their YouSearch bloom filters. Define R_q to be the set of providers that are members of groups deemed relevant by the constructed index. Then, *loss in recall* for q due to the constructed index can be quantified as $|S_q - R_q|/|S_q|$. The average loss in recall for a query set is simply the average of loss in recall for constituent queries.

Figure 11 plots size of privacy group c on X-axis against the average loss in recall observed on Y-axis. Note that the Y-axis is measured in percent (%) units. We observe that loss in recall is very small (less than 0.2%) and decreases with increasing accuracy ϵ of the index. With an epsilon setting of 0.001, the search is effectively complete. Note that there is no loss of recall for the cryptographic algorithm since it computes the exact group content vector.

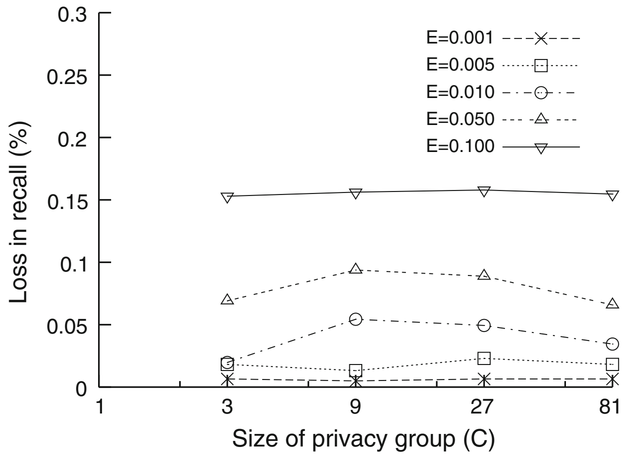

Fig. 11 Loss in recall

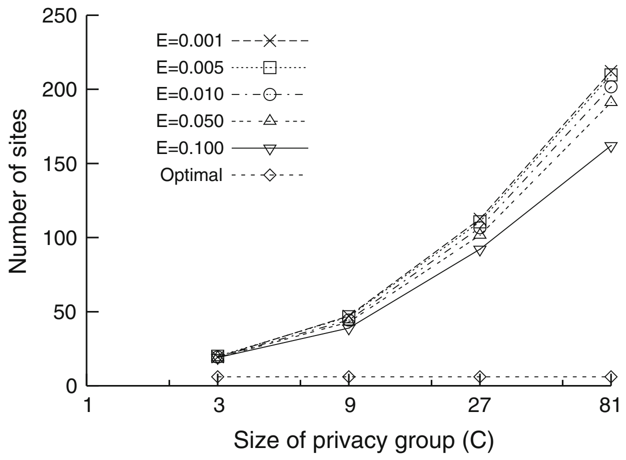

Fig. 12 Query processing cost

7.6 Query processing cost with PPI

We next show the effects on query performance in practice by running the indexing and search algorithm on real data obtained from a deployment of YouSearch within the IBM corporate intranet. A content provider in YouSearch constructs a bloom filter content vector with length $L = 64$ Kbits. The hash function H used for constructing the bloom filter is computed as follows. An MD5 hash is computed for each term t. The first 2 B of the 16 B long hash are used as the value for $H(t)$. The set of terms used to create the content vector are extracted from the filenames and pathnames of shared files and bodies of text and HTML shared files. The dataset used here was obtained from collecting all bloom filters from the set of peers actively participating in the search network at a specific point in time.

Figure 12 plots the average incurred cost observed over all queries. The X-axis plots size of privacy group c while Y-axis plots the average number of sites that evaluate a query. In the absence of privacy concerns, the query would be evaluated only at sites that had the corresponding bit positions in their bloom filters set to 1 (indicated as the *Optimal* curve). However, the index causes *all* c sites in a c sized privacy group to be searched incurring a penalty in query processing cost. We observe that increasing c leads to similar increase in query processing costs. When averaged over all queries in our test set, the performance penalty (and privacy benefit) is roughly $\frac{2}{3} \times c$. An increase in ϵ (decrease in accuracy) also results in decreased processing cost. However, such decreases are merely an artifact of reduced recall due to the group content vector inaccuracies implied by larger settings of ϵ.

8 Related work

Researchers have identified the importance of preserving privacy during searching. The celebrated paper by Chor et al. [9] introduced the problem of *Private Information Retrieval*. A user wishes to privately retrieve the ith bit from a database, without revealing any information about i. Gertner et al. [20] introduced the stronger model of *Symmetrically Private Information Retrieval* which, in addition to maintaining the user's privacy, prevents the user from obtaining any information other than a single bit of the data. The privacy of the user is defined in an information-theoretic setting, which makes it hard to find practical and efficient schemes. SPIR schemes often require multiple non-colluding servers, consume large amounts of bandwidth and do not support keyword searching. Although our scheme does not provide information-theoretic security bounds, it has low computational and communication complexity while providing probabilistic privacy guarantees.

Statistical database research strives to provide aggregate information without compromising sensitive information about individual records [1]. Secure databases research attempts to prevent unauthorized access to records in the database [29]. Popular search solutions build inverted indexes on shared content [8,26,31]. All of these solutions assume a trusted centralized server. We believe that autonomous content providers will find it difficult to form a consensus on such a trusted host.

Researchers have investigated the problem of running queries over encrypted data at an untrusted server [27,35]. The schemes require the searcher to know a secret key with which content accessible to the searcher is encrypted. The searcher now has to explicitly maintain secret keys for each provider she has access to. As far as we know, no previous work has defined PPI to enable global keyword searches.

In practice, people have preferred replacing privacy definitions with *anonymity* where the requirement is that the identity of the user ("Bob") be masked from an adversary ("Alice"). Low cost systems have been designed for various applications that involve building an "anonymous channel" that hides Bob from Alice. For example, Onion Routing [37], Crowds [33] and Tarzan [17] allow the source of a message to remain anonymous. Freenet [18] and FreeHaven [14] ensure provider privacy for file sharing. Freenet supports searches *without* access control over shared content. Freehaven does not support searches but preserves the anonymity of readers. It is not obvious how these schemes can be adapted to enable content privacy while searching access-controlled content.

Researchers in the field of secure multi-party computation [24] have developed theoretical methods for securely computing functions over private information distributed across any number of hosts. Recent work in this area has focused on developing more efficient schemes for specific functions. A bloom filter scheme using encryption for hash functions has recently been proposed by Goh [22]. This allows searching on encrypted data and is orthogonal to our work. Secure co-processors [15] can also be used for improving the privacy of PPI construction, should one with sufficient processing power, storage and bandwidth be available for use by participating providers.

In concurrent work, Bellovin and Cheswick [4] present a search scheme based on Bloom filters and Pohlig-Hellman encryption to provide "query privacy" over remote data. The index host can transform a user's search queries to a form suitable for querying a remote host, in such a way that neither the index host nor the remote host can deduce the original query. This does not address the problem of allowing multiple untrusting parties to create a single privacy preserving index over local access controlled documents. Our work maintains this "content privacy" and defines a data structure (PPI) that can be achieved using Bloom filters. Schadow et al. [34] also discuss a similar scheme to perform distributed queries via record linkage using hash values. The main contribution is to propose an algorithm for distributed joins using Bloom filters for hashed record linkage. While the model is somewhat similar, as a mediator is used to merge queries, our work is orthogonal to this, and also deals with queries at a higher level. Rather than record linkage, we are more interested in enabling search. Once the centralized index is constructed, no further aggregation is necessary by the central party.

9 Conclusions

We have addressed the challenge of providing privacy-preserving search over distributed access-controlled content. Conventional inverted indexes represent an indexed document in its virtual entirety. The trust and security thus required of any host providing such an index over access-controlled content is enormous. In fact, as the number of participating information providers grows, this required level of trust quickly becomes impractical. Our solution eliminates entirely the need for such a trusted indexing host through the use of a PPI.

We defined and analyzed a PPI and presented two different algorithms for constructing a PPI. The randomized algorithm is extremely efficient but has lower privacy guarantees and is susceptible to collusion. The cryptographic method is significantly more secure, but is also significantly slower. This provides an interesting tradeoff, and allows a user to choose based on their situational requirements. We showed that the index, once constructed is strongly resilient to privacy breaches even against *colluding* adversaries. Experiments on a real-life dataset validate performance of our scheme.

Our solution enables content providers to maintain complete control in defining access groups over their content and ensuring its compliance. Moreover, implementors can use the size of privacy groups as a tuning knob to balance privacy and efficiency concerns for their particular domains.

One interesting avenue for future work is that of efficiently handling index updates. Currently, we handle index updates simply by rerunning the basic index construction protocol. Since this is quite efficient (especially for the randomized algorithm), this is ok. But in many situations, index updates can be quite frequent and more efficient solutions would be highly desirable. Secondly, our base assumption is that the list of providers does not change very frequently. If this is also subject to frequent change (as in BitTorrent like applications), a more efficient solution must be found. Another issue is that of index efficiency. While our solution meets privacy requirements, and provides reasonable efficiency, ideally we would like to create a solution that guarantees efficiency as well as privacy. This is another avenue for future research.

References

1. Adam, N.R., Wortman, J.C.: Security-control methods for statistical databases. ACM Comput. Surv. **21**(4), 515–556 (1989)
2. Bawa, M., Bayardo, R.J. Jr., Rajagopalan, S., Shekita, E.J.: Make it fresh, make it quick—searching a networks of personal webservers. In: Proceedings of the Conference on World Wide Web (WWW) (2003)
3. Bayardo, R.J. Jr., Agrawal, R., Gruhl, D., Somani, A.: YouServ: A web-hosting and content sharing tool for the masses. In: Proceedings of the 11th International Conference on World Wide Web, Honolulu, Hawaii, USA, pp. 345–354 (2002)
4. Bellovin, S., Cheswick, W.: Privacy-enhanced searches using encrypted bloom filters. (2004)
5. Benaloh, J.C.: Secret sharing homomorphisms: keeping shares of a secret secret. In: Proceedings of the Advances in Cryptography (CRYPTO) (1986)
6. Bloom, B.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
7. Blum, M., Goldwasser, S.: An efficient probabilistic public-key encryption that hides all partial information. In: Proceedings of Advances in Cryptology (CRYPTO) (1984)
8. Brin, S., Page, L.: Anatomy of a large-scale hypertextual web search engine. In: Proceedings of the Conference on World Wide Web (WWW) (1998)
9. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: Proceedings of the Conference on Foundations of Computer Science (FOCS) (1995)
10. Choudhury, A., Maxemchuk, N., Paul, S., Schulzrinne, H.: Copyright protection for electronic publishing over computer networks. AT&T Bell Laboratories Technical Report (1994)
11. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. Advances in Cryptology—EUROCRYPT 2001. Lecture Notes in Computer Science, vol. 2045, pp. 280–300. Springer, London (2001)
12. Damgård, I.B., Jurik, M.J.: Efficient protocols based on probabilistic encryption using composite degree residue classes. Technical Report RS-00-5, BRICS (2000)
13. Dierks, T., Allen, C.: The tls protocol. RFC 2246, Standards Track, Network Working Group (1999)
14. Dingledine, R., Freedman, M.J., Molnar, D.: The free haven project: Distributed anonymous storage service. In: Proceedings of the Workshop on Design Issues in Anonymity and Unobservability (2000)
15. Dyer, J., Lindemann, M., Perez, R., Sailer, R., Smith, S.W., van Doorn, L., Weingart, S.: Building the ibm 4758 secure coprocessor. IEEE Comput. **34**, 57–66 (2001)
16. Fouque, P.-A., Poupard, G., Stern, J.: Sharing decryption in the context of voting or lotteries. In: Proceedings of the International Conference on Financial Cryptography (2001)
17. Freedman, M.J., Morris, R.: Tarzan: A peer-to-peer anonymizing network layer. In: Proceedings of the Conference on Computer and Communications Security (2002)
18. The freenet project (http://freenetproject.org)
19. Frier, A., Karlton, P., Kocher, P.: The ssl 3.0 protocol. Netscape Corp., 1996
20. Gertner, Y., Ishai, Y., Kushilevitz, E., Malkin, T.: Protecting data privacy in private information retrieval schemes. In: Proceedings of the Symposium on Theory of Computation (STOC) (1998)
21. The gnutella network (http://gnutella.com)
22. Goh, E.-J.: Secure indexes. Cryptology ePrint Archive, Report 2003/216 (2003)
23. Goldreich, O.: The Foundations of Cryptography, vol. 2. General Cryptographic Protocols, chap. Cambridge University Press, London (2004)
24. Goldwasser, S.: Multi-party computations: past and present. In: Proceedings of the Symposium on Principles of Distributed Computing (1997)
25. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: Proceedings of the Symposium on Theory of Computing (STOC) (1985)
26. Gravano, L., Garcia-Molina, H., Tomasic, A.: Gloss: Text source discovery over the internet. ACM Trans. Database Syst. **24**(2), 229–264 (1999)
27. Hacigumus, H., Iyer, B., Li, C., Mehrotra, S.: Executing sql over encrypted data in the database-service-provider model. In: Proceedings of the Conference on Management of Data (SIGMOD) (2002)
28. The kazaa media network (http://www.kazaa.com)
29. Landwehr, C.: Formal models of computer security. ACM Comput. Surv. **13**(3), 247–278 (1981)
30. Naccache, D., Stern, J.: A new public key cryptosystem based on higher residues. In: Proceedings of the Conference on Computer and Communications Security (CCS) (1998)
31. Napster file-sharing (http://www.napster.com)
32. Okamoto, T., Uchiyama, S.: A new public-key cryptosystem as secure as factoring. In: Proceedings of Advances in Cryptology (Eurocrypt) (1998)

33. Reiter, M.K., Rubin, A.D.: Crowds: Anonymity for Web transactions. ACM Trans. Inform. Syst. Secur. **1**(1), 66–92 (1998)
34. Schadow, G., Grannis, S.J., McDonald, C.J.: Privacy-preserving distributed queries for a clinical case research network. In: Proceedings of the IEEE International Conference on Data Mining; Workshop on Privacy, Security, and Data Mining (2002)
35. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Proceedings of the (IEEE) Symposium on Security and Privacy (2000)
36. Stern, J.P.: A new and efficient all or nothing disclosure of secrets protocol. In: Kazuo, O., Dingyi, P. (eds.) Advances in Cryptology–ASIACRYPT'98, number 1514 in Lecture Notes in Computer Science (1998)
37. Syverson, P., Goldschlag, D.M., Reed, M.G.: Anonymous connections and onion routing. In: Proceedings of the IEEE Symposium on Security and Privacy (1997)