PEAS: Private, Efficient and Accurate Web Search

Albin Petit, Thomas Cerqueus, Sonia Ben Mokhtar, Lionel Brunie, Harald Kosch

PEAS: Private, Efficient and Accurate Web Search

Albin Petit, Thomas Cerqueus,
Sonia Ben Mokhtar, and Lionel Brunie
Université de Lyon, CNRS
INSA-Lyon, LIRIS, UMR5205, F-69621, France
Email: firstname.lastname@insa-lyon.fr

Harald Kosch
Universität Passau
Innstrasse 43, 94032 Passau, Germany
Email: harald.kosch@uni-passau.de

***Abstract*—Accounting for the large number of queries sent by users to search engines on a daily basis, the latter are likely to learn and possibly leak sensitive information about individual users. To deal with this issue, several solutions have been proposed to query search engines in a privacy preserving way. A first category of solutions aim to hide users' identities, thus enforcing *unlinkability* between a query and the identity of its originating user. A second category of approaches aims to obfuscate the content of users' queries, or at generating fake queries in order to blur user profiles, thus enforcing *indistinguishability* between them. In this paper we propose PEAS, a new protocol for private Web search. PEAS combines a new *efficient* unlinkability protocol with a new *accurate* indistinguishability protocol. Experiments conducted using a real dataset of search logs show that compared to state-of-the-art approaches, PEAS decreases by up to 81.9% the number of queries linked to their original requesters. Furthermore, PEAS is accurate as it allows users to retrieve up to 95.3% of the results they would obtain using search engines in an unprotected way.**

I. Introduction

Querying search engines (e.g., Google, Bing, Yahoo!) is by far the most frequent activity performed by online users for efficiently retrieving content from the tremendously increasing amount of data populating the Web. As a direct consequence, search engines are likely to gather, store and possibly leak sensitive information about individual users (e.g., interests, political and religious orientations, health condition). In this context, developing solutions to enable private Web search is becoming increasingly important. A lot of effort has been invested towards this direction. A first class of techniques, called *unlinkability*, aims to hide the identity of the requester to the search engine. Most of these approaches rely on anonymous communication protocols (e.g., Onion Routing [1], Tor [2], Dissent [3] and RAC [4]). However, if the search engine has access to external knowledge (e.g., user profiles collected from the history of past queries), these solutions do not resist de-anonymization attacks [5]. Furthermore, most of these solutions rely on heavy cryptographic mechanisms or on all-to-all communications, which makes them hardly usable for highly interactive tasks. A second class of techniques, called *indistinguishability*, proposes to obfuscate user queries or user profiles in order to make it difficult to distinguish users' real interests from fake ones. This is usually done by altering the original query (e.g., adding extra keywords) or by generating covert queries and sending them on behalf of the user [6]–[9]. However, these solutions degrade the accuracy of the results returned to the user or overload the network with a large number of fake queries.

As most of these solutions (e.g., Tor [2], TrackMeNot [6], GooPIR [7]) suffer from a de-anonymization attacks (i.e., machine learning attack [5] or SimAttack, a new attack presented later in this paper), a straightforward solution may consist in building a protocol combining unlinkability and indistinguishability solutions. Nevertheless, we show in this paper that running GooPIR on top of an unlinkability solution still fails to protect users in a satisfactory manner as 30% of the queries are still linked back to their corresponding user profiles.

To address this problem, we propose PEAS, a new protocol for enforcing private Web search. Specifically, PEAS combines a new efficient unlinkability protocol with a new accurate indistinguishability solution. To enforce indistinguishability, PEAS aggregates each user query with k fake queries using a logical OR operator. These requests are then sent to the search engine through a privacy-preserving proxy, which enforces unlinkability. A key feature of our approach, is that fake queries are constructed using a *group profile*. A group profile is an aggregation from past queries of a community of users and is published in a privacy-preserving way by the privacy proxy. Specifically, this group profile allows building fake queries that point at k disjoint clusters of users also built in a privacy-preserving way by the privacy proxy. Furthermore, fake queries have the same structure as the user's query (number of keywords, frequency of co-occurring terms etc.), which makes them hardly identifiable (as fake) by an adversary.

We assess the effectiveness of PEAS both theoretically and practically. Specifically, we prove that our privacy proxy guarantees unlinkability between user queries and their respective profiles using ProVerif [10]. Furthermore, we study the indistinguishability between fake queries and real ones. Our performance evaluation carried out using a real dataset of search queries (i.e., the AOL dataset [11]), shows that PEAS decreases by 81.9% the number of queries linked by the adversary to their initial requester compared to GooPIR over an unlinkability solution. Besides, PEAS accurately retrieves the results of the original queries, as for 95% of the queries, more than 80% of the expected results are retrieved. Finally, the performance of our privacy proxy clearly outperforms its natural competitor, i.e., the Onion Routing protocol as its round trip time is 6 times lower and its peak throughput is

3 times higher.

The remaining of the paper is organized as follows. In Section II, we present the related work. In Section III, we present an overview of the PEAS architecture, which is described in more details in Section IV. In Section V, we discuss the security of PEAS. In Section VI, we present the performance evaluation of PEAS. Section VII concludes the paper and discusses our future directions.

II. Related Work

The literature contains many solutions to protect users during Web search. These solutions can be classified in three categories: (*i*) systems guaranteeing *unlinkability* between users and their queries, (*ii*) systems guaranteeing *indistinguishability* of the users' queries, and (*iii*) systems implementing a Private Information Retrieval (PIR) scheme. We present these categories in the rest of this section.

A. Unlinkability Solutions

Unlinkability techniques aim at protecting users' privacy by hiding their identity (e.g., their IP addresses). The most naive solution in this category, consists in using a trusted proxy that forwards user queries to the search engine on behalf of the user [12]. As such, from the search engine's perspective, all the queries are issued by the same user (i.e., the proxy). However, this only shifts the problem from the search engine side to the proxy side, as the proxy is able to collect and possibly leak information about users. Therefore, more robust solutions such as anonymous communication protocols have been devised (e.g., Onion Routing [1], Tor [2], Dissent [3], RAC [4]). Nevertheless, these protocols have to perform a significant number of cryptographic operations, and some of them (e.g., RAC and Dissent) generate a lot of traffic on the network, as they heavily rely on all-to-all communication between the users participating in the protocol.

Besides anonymous protocols, other techniques have been proposed to ensure unlinkability. For instance, in [13] and [14], users exchange their queries with each other: the queries issued by a user are forwarded to another user that eventually sends the query to the search engine. Consequently, this misleads the search engine, as it can not infer the identity of the original requester. However, these approaches increase the latency perceived by the users, as the users need to communicate with each other to organize the swapping of their queries.

B. Indistinguishability Solutions

This category of approaches proposes to obfuscate user profiles to make them inaccurate. As a result, the search engine cannot distinguish between the real interests of a user and fake ones. To do so, usual solutions periodically send fake queries on behalf of the user. For instance, TrackMeNot (TMN) [6] uses RSS feeds to create fake queries. However, as these RSS feeds are statically chosen[1] or set up manually by the user, the generated queries might not be relevant to cover users' interests. Moreover, [15] shows that a clustering attack over small time frames enables an adversary to filter out fake queries. Plausibly Deniable Search (PDS) [8] adopts a similar approach: it generates from a set of documents k plausibly deniable queries similar to past user queries but on unrelated topics. Optimized Query Forgery (OQF) [16] uses the Kullback-Leiber divergence between the user profile and the population distribution to provide a theoretical method to generate new queries. Noise Injection for Search Privacy Protection (NISSP) [17] (similar approach to OQF) gives a theoretical analysis on the generation of the optimal fake queries. These four solutions (TMN, PDS, OQF and NISSP) tend to overload the network with additional traffic by generating a large number of fake queries.

Other techniques proposed in the literature, alter the initial query instead of covering it. For instance, GooPIR [7] aggregates $(k-1)$ random queries (based on a dictionary and on keywords frequency) to the initial query. This ensures that search engines can guess the correct query with a probability equal to $1/k$. However, we will show in Section VI that if the search engine has already a profile of the user, GooPIR is not able to effectively protect her query. Another technique called Query Scrambler [18] hides the initial query by sending a set of new queries generated by generalizing concepts of the initial query. It then tries to retrieve the original results (i.e., the results corresponding to the initial query) by filtering the union of results corresponding to the generated queries. However, this solution has a low accuracy, as results corresponding to the set of new queries do not contain all the results of the initial query. Finally, a recent approach, DisPA [19], tries to find a trade-off between user privacy and user profile exploitation. To do so, DisPa misleads the search engine about the identity of the user using multiple HTTP cookies. Consequently, the search engine aggregates user's queries in multiple user profiles. However, this approach relies on the strong assumption that the search engine identifies users with their cookies. Besides, the aim of this approach is to reduce (and not prevent) the disclosure of information and thus a sufficient protection of users' privacy cannot be guaranteed.

C. Private Information Retrieval

Private Information Retrieval (PIR) in the context of Web search prevents the Web search engine from knowing which information users are looking for. For instance, the system presented in [9] breaks down the query in multiple buckets of words. Using homographic encryption, it succeeds to retrieve search results without revealing the initial query. However, due to the homographic encryption, this solution is computationally costly. Besides, it requires a change on the search engine side to implement a recommendation algorithm that works with homographic encryption.

D. Discussion

As shown above, none of the aforementioned solutions are satisfactory to query search engines in privacy-preserving way as most of them suffer from de-anonymization attacks. We

[1] by default: `cnn.com`, `nytimes.com`, `msnbc.com` and `theregister.co.uk`

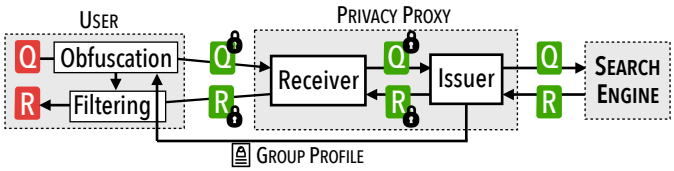


Fig. 1. Overview of our countermeasure.

present in the rest of this paper, PEAS, a new private Web search protocol which combines a new unlinkability protocol with a new indistinguishability solution.

III. PEAS in a Nutshell

As mentioned in section II, a simple solution to ensure unlinkability is to use a proxy that sends requests on behalf of the user. However, this only shifts the problem from the search engine side to the proxy side, as the proxy is able to collect and possibly leak information about the users. This is why, the first part of our contribution is a privacy-preserving proxy that ensures unlinkability between a user identity and her queries. This is done by distributing these information in two non-colluding servers: the *receiver*, which manipulates only user identities and the *issuer*, which manipulates only anonymized queries. Figure 1 shows an overview of PEAS. The receiver receives a message from the client, which contains a ciphered request (only decipherable by the issuer). It then assigns an identifier to this message and forwards it to the issuer. Then, the issuer deciphers it and forwards the query to the search engine. In this protocol, the receiver knows the identity of the requester without learning her interests, while the issuer accesses queries without knowing their provenance. Upon receiving a response from the search engine, the issuer ciphers the results and forwards them along with the identifier received with the query to the receiver. Using this identifier, the receiver retrieves the initial user identity and forwards her the message. Finally, the user deciphers the message and retrieves the results. Section IV-B further describes this protocol.

Nevertheless, it has been shown in the literature [5], unlinkability solutions are not enough to protect users' privacy, as a significant proportion of queries can be de-anonymized. To solve this issue, we propose to combine the privacy proxy with an obfuscation mechanism. This mechanism combines user queries with multiple fake queries using the logical OR operator. Fake queries aims to mislead the adversary about the original query. Nevertheless, this approach can be inefficient if fake queries are not generated appropriately. To overcome this limitation, we identify three requirements that the generation of fake queries needs to satisfy in order to produce realistic fake queries:

Requirement 1: Fake queries shall have similar structural properties as the initial one (e.g., number of keywords and usage frequency). We describe how we satisfy this requirement in Section IV-C.

Requirement 2: Fake queries shall be built from past queries. To address this key requirement, we propose to generate queries based on an aggregated history of past requests gathered from users of our system. This information, called the group profile, is published by the privacy proxy as shown in Figure 1. However, publishing users' past queries in a group profile may lead to information leakage. Consequently, it has to be done in privacy-preserving way. We describe how we deal with this challenge in Section IV-C.

Requirement 3: Fake queries shall point at different sets of users, otherwise, the adversary can easily filter them out. That is why, the group profile contains sets of aggregated queries built from disjoint users. Each set of aggregated queries is used to generate one fake query at a time. A detailed description of how the group profile is structured is presented in Section IV-C.

Finally, obfuscating user queries may alter the quality of the results received by the users. To overcome this issue, PEAS filters out the results to remove irrelevant answers (i.e., answers corresponding to fake queries). This filtering step is described in Section IV-D.

IV. PEAS Detailed Description

In this section, we present the detailed description of PEAS. After presenting our assumptions in Section IV-A, we describe the privacy proxy in Section IV-B. We then introduce the obfuscation mechanism followed by the group profile publication protocol in Section IV-C. Finally, we present the filtering mechanism in Section IV-D and discuss the large-scale deployment of PEAS in Section IV-E.

A. Assumptions

We assume that the two servers composing the privacy proxy are honest-but-curious and do not collude. It means that these servers must follow the protocol (i.e., they do not forge, modify, replay or drop messages) and do not exchange extra information with each other. However, they can locally monitor information exchanged in the protocol and exploit this knowledge with other sources of information. We also assume that the search engine has obtained external knowledge about the users and has therefore built a user profile for each of them. Indeed, we assume that users have been using the search engine in the past without any privacy-preserving protection.

B. Privacy Proxy

The privacy proxy aims to ensure unlinkability between the user and her queries. To achieve this goal and as we want an efficient protocol, the privacy proxy is split into two separated servers: the receiver knows the identity of the users (e.g., their IP address) while the issuer knows the content of their queries. Figure 2 shows how our protocol achieves to hide information to one component or the other: when issuing a query q, the user creates a new symmetric cryptographic key K_q (block A in the Figure). Then, using the RSA encryption algorithm and the issuer's public key pk_I, she ciphers the aggregation of her query q and her key K_q (block B). The cipher-text $E(q \oplus K_q)_{pk_I}$ is then sent to the receiver (message ❶). Since the receiver does not know the issuer's private key, it is not able to access the content of the initial query. As soon as the

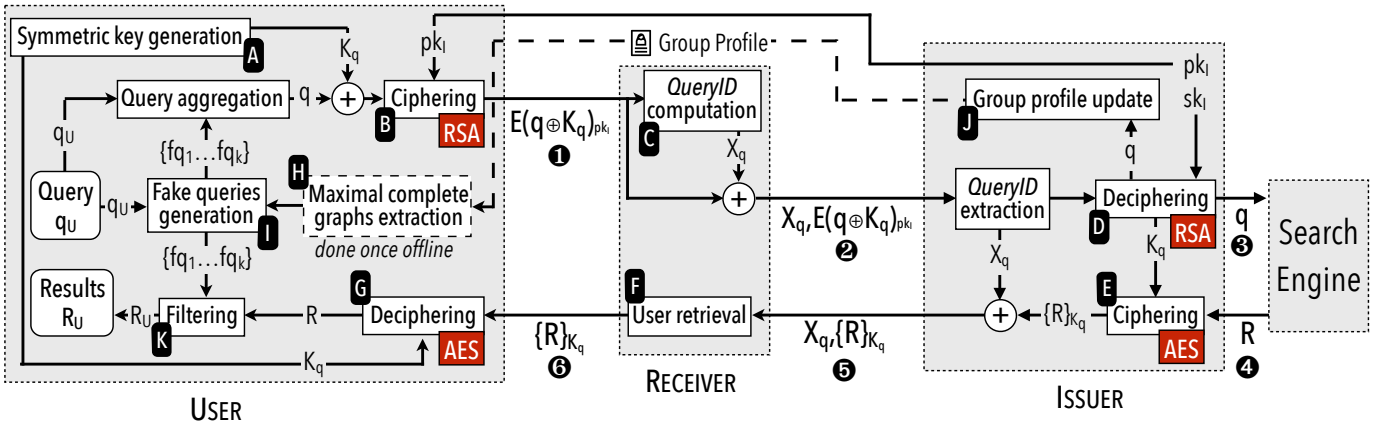


Fig. 2. Sequence diagram of PEAS.

receiver receives the message, it assigns a unique identifier X_q to the query (block C). Then, it forwards the message along with the identifier X_q to the issuer (message ❷). The issuer receives the cipher-text from the receiver so it has no information about the user identity. It uses its private key to decipher the message (block D) and retrieve the plaintext (concatenation of the user's query q and the user's symmetric key K_q). Then submits the initial query q to the search engine (message ❸). Upon receiving the results (message ❹), the issuer uses the user's symmetric key K_q to encrypt with the AES encryption algorithm the search engine's answer R (block E) and return the new cipher-text $\{R\}_{K_q}$ along with the user's identifier X_q to the receiver (message ❺). Finally, the receiver retrieves the requester identity using the identifier (block F) and sends the encrypted results $\{R\}_{K_q}$ to the initial user (message ❻). The user then uses the cryptographic key K_q to retrieve the results of her query (block G).

For privacy reasons, the user needs to generate a new cryptographic key K_q every time she wants to issue a query. Otherwise, the key might be used by the issuer to link a set of consecutive queries issued by the same user and ultimately recover her identity (as shown in [20]).

C. Obfuscation Mechanism

An attack presented in [5] shows that hiding users identity (i.e., using the privacy proxy alone) is not sufficient to protect their privacy. Therefore, it is necessary to make queries unlinkable to the user profiles. That is why, we propose a new query obfuscation mechanism. Nevertheless, we advocate that as the user wants to retrieve the results that she would get without any protection mechanisms, the obfuscated query must contain the keywords of initial query. Consequently, our obfuscation mechanism aggregates the initial query with k fake queries separated with logical OR operators. This allows, in a post processing phase, to retrieve results of the initial query by filtering out the irrelevant results introduced by the fake queries.

1) Fake Queries Requirements: A key challenge is to generate fake queries that are not distinguishable from the initial query. They must satisfy the three requirements defined in Section III. Therefore, our idea consists in generating k queries that (*i*) have the same number of keywords than the initial queries, (*ii*) have a similar usage frequency that the initial queries, and (*iii*) have been issued by more than 1 user. Besides, to mislead the adversary about the requester identity, these fake queries must not contain keywords that have already been used by the requester. In other words, the generated fake queries must be far from the user profile and close to other users.

TABLE I
EXAMPLE OF A CO-OCCURRENCE MATRIX INCLUDED IN A GROUP PROFILE.

	HIV	treatment	depression	test	symptom
HIV	-	1	0	10	11
treatment	-	-	5	0	4
depression	-	-	-	17	13
test	-	-	-	-	0
symptoms	-	-	-	-	-

2) Group Profile Exploitation: In order to generate realistic fake queries, we propose to exploit two types of information: the past queries of the user, and the past queries of other users. We consider that each user locally stores a history of her queries. Besides, as it has no direct access to the past queries of other users, the user relies on a group profile periodically published by the privacy proxy. This group profile is structured in multiple clusters of disjoint users. The past queries of a given user are aggregated in one cluster. Each cluster is associated with a co-occurrence matrix M such that the matrix's axes represent the vocabulary (i.e., keywords already used in past queries). Each element $M(a, b)$ of the matrix represents the frequency of occurrence of the keywords a and b. The construction of the co-occurrence matrix for each cluster is performed by the issuer, as it is the only entity that has access to the content of the queries. Table I gives an example of such a co-occurrence matrix. The elements corresponding to the pair ("HIV", "treatment") contain 1 because one query containing "HIV" and "treatment" has been issued in the past.

By construction of the co-occurrence matrix (i.e., aggregation of bi-occurrence frequencies), users have only access to

an approximation of past queries. To generate fake queries using this matrix, users transpose a co-occurrence matrix to a co-occurrence graph, and compute all maximal complete graphs, also called *maximal cliques* (block H in Figure 2). The completeness property is a necessary condition for already formulated queries, i.e., already formulated queries are complete graphs of keywords. Furthermore, the maximality property aims to maximize the probability of selecting complete queries and not a subset of past queries. The computation of all the maximal cliques of a graph is a NP-complete problem, which we overcome using the Bron-Kerbosch algorithm [21]. Besides, this computation does not impact the latency as it is done offline once.

Algorithm 1: Generation of fake queries.

input: q_u : original query,
k : number of fake queries to create,
P_u : profile of a user u,
F_{width} : frequency interval,
M : sum of all co-occurrence matrices,
G : array of k cells containing all maximal cliques of a given cluster.

1 $FQ \leftarrow \emptyset$;
2 $\Phi_{q_u} \leftarrow \min_{(a,b)\in q_u^2, a\neq b} M(a,b)$;
3 $F_{min} \leftarrow \texttt{random}(\Phi_{q_u} - (F_{width}/2), \Phi_{q_u})$;
4 $F_{max} \leftarrow F_{min} + F_{width}$;
5 **while** $|FQ| < k$ **do**
6 $\quad g \leftarrow G[|FQ|]$;
7 $\quad fq \leftarrow \texttt{generateFakeQuery}(g, |q_u|, F_{min}, F_{max})$;
8 $\quad$ **if** $(fq \cap q_u = \emptyset \wedge fq \cap P_u = \emptyset)$ **then**
9 $\quad\quad FQ \leftarrow FQ \cup \{fq\}$
10 **return** FQ;

3) Fake Queries Generation: The approach proposed to generate fake queries (block I in Figure 2) is presented in Algorithm 1. Input parameters include F_{width}, which represents the width of the frequency interval fake queries have to belong to. The value Φ_{q_u} represents the normalized usage frequency of the original query q_u (line 2). An interval $[F_{min}, F_{max}]$ containing Φ_{q_u} is then defined (lines 3 and 4). The method `random` allows to get an interval which is not necessarily centered on Φ_{q_u}. The method $\texttt{generateFakeQuery}(g, n, x, y)$ is used to generate a query fq such that $|fq| = n$ and $\Phi_{fq} \in [x, y]$ (line 7). We let g be the set of maximal cliques to extract words to create fq. This maximizes the probability that the selected words already appeared in a same query. Thus, the generated query is likely to be close to a user profile. When a query is generated, n words are picked in a graph of g. This subgraph of size n is either (*i*) a maximal clique, (*ii*) a non-maximal clique, or (*iii*) a non-complete graph. The choice is made randomly to mislead potential adversaries. Indeed, an adversary cannot use the frequency nor the number of keywords to determine if a query was issued by a user, or generated by the obfuscation mechanism. Notice that a generated query is included in the set of fake queries if and only if it does not contain words that also appear in the user profile P_u and in the original query q_u (lines 8 and 9). If this condition is not verified, the noise introduced in the results of the obfuscated query might be difficult to remove.

4) Privacy-Preserving Group Profile Publication: To obfuscate their queries, users need to have access to the group profile. As mentioned in the previous subsection, the issuer computes and releases the group profile periodically (block J in Figure 2). The group profile is composed of $k + 1$ co-occurrence matrices (i.e., clusters) in which the issuer aggregates all the queries. Nevertheless, the issuer has no information about which query, among the $k + 1$ queries aggregated with the logical OR operator, belongs to which cluster. To overcome this issue in a privacy-preserving way, the user gives this information by aggregating her query and the fake queries in the same order as the clusters (i.e., the first query belongs to cluster 1, the second query belongs to cluster 2, etc.). Besides, to generate the k fake queries and to aggregate queries in the correct order, the user needs to know which k clusters (among the $k + 1$) she needs to use and which cluster is associated to her real queries. This is done by sending once a special query to the receiver. When the receiver receives this special query, it randomly assigns the user to one of the $k + 1$ clusters. Finally, it answers the user with her cluster number.

D. Filtering Irrelevant Results

Algorithm 2: Results filtering.

input: q_u : original query,
$FQ = \{fq_1, \ldots, fq_k\}$: set of fake queries,
R : set of results for $q_u \vee fq_1 \vee \cdots \vee fq_k$.

1 $\bar{R} \leftarrow \emptyset$;
2 $q^+ \leftarrow \{q_u, fq_1, \ldots, fq_k\}$;
3 **for** $r \in R$ **do**
4 $\quad$ **for** $q_i \in q^+$ **do**
5 $\quad\quad score[q_i] \leftarrow \texttt{nbCommonWords}(q_i, \texttt{title}(r))$
6 $\quad\quad\quad + \texttt{nbCommonWords}(q_i, \texttt{desc}(r))$
7 $\quad\quad\quad + \texttt{nbCommonWords}(q_i, \texttt{url}(r))$;
8 $\quad$ **if** $score[q_u] = \max_{q_i \in q^+} score[q_i]$ **then**
9 $\quad\quad \bar{R} \leftarrow \bar{R} \cup \{r\}$;
10 **return** $\bar{R}$;

The filtering aims to remove the irrelevant results introduced by the fake queries (block K in Figure 2). Indeed, search engines answer with a mix of results corresponding to $(k+1)$ queries (k fake queries and the initial one). As we want the user to retrieve results that she would get without any protections (i.e., corresponding to the initial query), it is needed to filter out all the results. This filtering is done on the client side, as the user is the only one knowing which one is the initial query. Algorithm 2 describes the filtering process. For each result r from the result set, the algorithm determines if it has been retrieved because it relates to the initial query or to a fake query. A similarity score is assigned to each query (lines 5

to 7). It considers meta-data (typically the title, description and URL). The method `nbCommonWords(`q, e`)` relies on the number of common words between a query q and an element e. It is defined as $|q \cap e| + |q \cap HW_e|$, where HW_e is the set of highlighted words in e (e.g., the words tagged in bold in the description). A result r is considered to relate to the initial query if and only if the initial query has the largest score (lines 8 and 9).

E. Discussion on the Deployment of PEAS

The deployment of PEAS raises two main issues. On the one hand, at the cold start of our system, no group profile is available to generate fake queries. A possible solution to deal with this issue consists in creating a group profile with dummy queries extracted from external sources (e.g., AOL search logs [11], Google Trends [22]). By doing so, the privacy risks for the users would be similar to those of our competitors (e.g., GooPIR), which rely on similar data to generate their fake queries. Yet, PEAS would rapidly outperform our competitors as the group profile gets constructed.

On the other hand, our architecture has to support a large number of users. Towards this purpose, we envision a distributed deployment of PEAS *à la* Tor, as this would help to distribute the load on multiple servers. Indeed, our architecture can be composed of multiples receivers and issuers. Thus, when a user wants to use the PEAS architecture, she has to select a receiver and an issuer for each query. For better performance and security, this choice may take into account the load of servers or the trust that users have on servers. Besides, in this distributed scenario each issuer publishes its own group profile along with its public key. Consequently, to generate fake queries, the user exploits the group profile published by the issuer that she plans to use.

V. Security Analysis

This section presents the elements that may jeopardize the system and the guarantees offered by PEAS.

A. On the Robustness of the Unlinkability Guarantee by the Privacy Proxy

We used ProVerif [10] to formally prove the unlinkability property of the privacy proxy protocol. ProVerif is a well-known tool to perform security analysis of cryptographic protocols. Based on a formal model (e.g., π-calculus), it can verify several security properties.

We model our protocol according to the description given in Section IV. The 4 entities *user*, *receiver*, *issuer* and *search engine* are defined through subprocesses. We set up an adversary that tries to link together the query and the user by monitoring the network (i.e., public channel). Internally, ProVerif attempts to prove that a state in which the adversary links the query with the user is unreachable. However, ProVerif does not natively implement the notion of linkability. It can state that an adversary knows the query q and the user identity u, but it cannot determine if they are related (i.e., q was issued by u). To overcome this limitation, we created a constructor `LINK(`x`,`y`)` symbolizing that x and y are linked, and we made this constructor transitive with a destructor `INFER`.

By using the full description accessible online[2], we formalized the privacy proxy protocol in type π-calculus. As indicated by the outcome of ProVerif, no attackers are able to link the query to its requester as the state cannot be reached. Indeed, an adversary that monitors the network (i.e., that follows the packets received and forwarded by the receiver) can in the worst case, only extract the same knowledge as a curious receiver. Therefore, we can deduce that the critical entity of our protocol is the issuer. Indeed, the unlinkability cannot be guaranteed if the receiver colludes with the adversary or the receiver.

B. On the Non-Collusion of the Receiver and the Issuer

The security of the privacy proxy relies on the assumption that the issuer and the receiver do not collude. This strong assumption is also made by other well-known anonymous or privacy-preserving protocols such as Tor or KOI [23]. In order to mitigate the risk of collusion, Tor enables users to use a higher number of relays. In our case, we prefer not to increase the number of intermediate servers, as we aim to keep a low overall latency. This choice is justified by the fact that queries sent through the privacy proxy are already obfuscated. Hence, a punctual leakage would not disclose too much information about the users especially in the distributed architecture where users would rely on different receivers and issuers to send their requests. Furthermore, similarly to KOI, one could imagine that institutions that fight for data privacy (e.g., Mozilla, academic institutions) might be willing to host a set of receivers and/or issuers.

C. On the Data Leakage Due to the Publication of the Group Profile

To decrease the risk of data leakage, we choose to publish information about all the users contributing to a group profile in the form of co-occurrence matrices. Indeed, this format aggregates data about all the users and thus gives no information about individual users. Besides, as the vectorial space is a set of unigrams (and not n-grams), the co-occurrence matrix contains only part of the initial information. For instance, if the adversary knows that a co-occurs with b, b co-occurs with c and c co-occurs with a, it cannot know whether the original query was "a b c" or whether it was a set of three independent queries "a b", "b c" and "c a". Moreover, choosing a low frequency of refreshing for the group profile (e.g., everyday) assures that the probability that only one user used the system is low. One could imagine a temporal attack in which the adversary studies the difference between two (or more) releases of the group profile. It would allow the adversary to retrieve information about past queries of multiple users. However, as the search engine is considered to be a potential adversary, we can state that it has already this knowledge and that consequently it does not obtain additional knowledge by performing this attack.

[2]Privacy Proxy ProVerif model: http://pastebin.com/5dVDEydC

D. On the Indistinguishability Between Initial Queries and Fake Queries

We generate fake queries that have equivalent properties to the initial query: they have the same usage frequency, the same number of keywords, and are built using co-occurence information taken from aggregated user queries. Consequently, an adversary cannot deduce by clustering (or applying an equivalent attack) which query is the initial one. Besides, the obfuscation mechanism ensures by design that each fake query refers to a different cluster of users. If it was not the case, this information could be used to identify fake queries, as by construction fake queries are different from the initial requester profile, the initial requester can necessarily be retrieved only once.

VI. PEAS Experimental Evaluation

This section presents the experimental evaluation of PEAS. We first describe the datasets we used in Section VI-A. Then we present the methodology and results of our evaluation, which considers three main dimensions: privacy, accuracy and performance, respectively described in Sections VI-B, VI-C and VI-D. Our evaluation draws the following conclusions: First, PEAS decreases by up to 81.9% the queries linked by an attacker to its originating user compared to GooPIR over an unlinkability solution. This value is even higher (up to 99%) if the attacker takes into account a confidence level on the results of its attack. Second, PEAS accurately retrieves a large percentage of the results of the original queries, as for 95% of the queries, 80% of the expected results are retrieved. This is twelve times higher than our competitor. Finally, the performance of our privacy proxy clearly outperforms its competitor, i.e., the Onion Routing protocol as its round trip time is 6 times lower and its peak throughput is 3 times higher.

A. Datasets

For these experiments, we consider queries from the AOL dataset [11]. This dataset contains approximately 21 millions queries formulated by 650,000 users over three months. We focus on the 300 most active users, as these users are the most vulnerable to re-identification attacks due to their profiles (which can be built with precision by the adversary). To remove robots and strange behavior, we only consider users that respect the two following criteria: (*i*) issued queries on a period of at least 61 days, i.e., there is at least 61 days between the first and the last query of the dataset (61 days represent 2/3 of the period of activity captured in the AOL dataset), and (*ii*) have been active for at least 45 days, i.e., they issued queries at least every second day. All together, the 300 most active users issued 343,548 queries (1,145 queries per user on average). We split this set of users into three datasets to cross-validate our results.

B. Privacy

In this section we evaluate PEAS from a privacy point of view and compare it with GooPIR over an unlinkability solution in order to have a fair comparison.

1) Methodology: To run these experiments, we used the two third of user queries to build user profiles, and try to de-anonymise the remaining one third of queries obfuscated with PEAS or GooPIR over an unlinkability solution. This de-anonymization is performed using two attacks: (*i*) an existing machine learning attack (ML attack) [5] and, (*ii*) SimAttack, a new attack based on a similarity metric between queries and user profiles.

Machine learning attack — We adapt the machine learning attack presented in [5] to de-anonymize obfuscated queries. We first build a model using Support Vector Machine algorithm on the two third of user queries. Then, we use this model to predict the user who issued a query. In our experiments, queries were obfuscated using k fake queries. Consequently, we evaluate each $k+1$ queries and consider the one with the higher probability as the real one, and the user predicted by the model as the real user.

SimAttack — We define SimAttack, a new attack to de-anonymize obfuscated queries. This attack aims at identifying both the initial requester and the initial query. SimAttack is based on a similarity metric $sim(q, P_u)$ that characterizes the proximity between a query q and a user profile P_u. This metric is computed as presented in Algorithm 3. It first computes the value $proj[q_i]$ corresponding to the cosine similarity between the query q and each query q_i already issued by the user (line 2). The projections $proj[i]$ are then ranked in ascending order (line 3). Finally, he similarity metric $sim(q, P_u)$ is computed as the exponential smoothing of these projections (lines 4 to 6). We empirically set the smoothing factor α to 0.6, as it maximizes the result of the attacks on the three datasets.

Algorithm 3: Similarity metric between a query and a user profile.

input: q : a query,
P_u : a user profile,
Q_u : queries issued by u,
W_u : set of words used in Q_u,
α : a smoothing factor.

1 **for** $q_i \in Q_u$ **do**
2 $\quad proj[i] \leftarrow \frac{1}{||q_i|| \cdot ||q||} \times \sum_{w \in W_u} q[w] \cdot q_i[w]$;
3 $proj \leftarrow sort(proj)$;
4 $sim \leftarrow proj[0]$;
5 **for** $i \in [\![1, |Q_u|]\!]$ **do**
6 $\quad sim \leftarrow \alpha \cdot proj[i] + (1-\alpha) \cdot sim$
7 **return** sim ;

For each query q_i, SimAttack (see Algorithm 4) retrieves the potential requester $id[i]$ by computing for each user profile P_{u_i} its similarity with the query q_i (lines 2 to 6). Then, it keeps queries that have for potential requester id a non-null value (lines 8 to 9). Finally, it retrieves queries a and b which have the highest similarities (lines 10 and 11) and it evaluates the difference between the similarity values of these two queries. To ensures a certain confidence in the results, if the difference

is greater than a threshold (set to 0.01 by default), it returns the pair $(q_a, id[a])$ corresponding to the initial query q_a and to the initial requester $id[a]$ (lines 12 and 13). Otherwise, the attack is unsuccessful.

Algorithm 4: SimAttack.

```
   input: q+ : a query,
          U : set of users
1  q' ← q0;
2  for qi ∈ q+ do
3      id[i] ← u0 ;
4      for ui ∈ U do
5          if sim(qi, P_ui) > sim(qi, P_id) then id[i] ← ui;
6      if sim(q, P_id) = 0 then id[i] ← null;
7  I ← [[0, k]];
8  for i ∈ [[0, k]] do
9      if id[i] = null then I ← I\{i}; continue ;
10 a ← index s.t. sim(qa, P_id[a]) is maximal over I;
11 b ← index s.t. sim(qb, P_id[b]) is maximal over I\{a};
12 if sim(qa, P_id[a]) − sim(qb, P_id[b]) > 0.01 then
13     return (qa, id[a]);
14 return ∅ ; // the attack is unsuccessful
```

2) De-anonymization Attack: Figure 3 shows the percentage of de-anonymized queries for both PEAS and GooPIR attacked by ML attack and SimAttack. The left bars of this figure (i.e., when k equals to 0), show results without any obfuscation mechanism; only unlinkability is used to protect user queries. Compared to this baseline, adding one fake query using PEAS decreases by half the number of linked queries (from 39.0% to 20.8%), while adding seven fake queries makes this number decrease by more than six times for both attacks (from 39.0% to 6.4%). From this experiment, we also note that PEAS clearly outperforms GooPIR over an unlinkability solution in all configurations. Indeed, PEAS protects at least 13.9% and 20.9% more queries than GooPIR for the configuration with 1 and 7 fake queries respectively.

The results obtained by PEAS can be explained by the fact that the more fake queries we create, the higher the probability that one of the fake queries, gets closer to a given user profile, than the original query is to its requester's profile. Note that, the obfuscation mechanism of PEAS does not protect all the queries as some of them are still linked to their originating user. We mitigate this result, by showing in the following section that the confidence obtained by the attacker about these queries is actually very low.

3) Confidence Level of the Adversary: The previous results do not take into consideration the confidence level that an adversary has in the outcome of its attack. Indeed, the adversary considers the query and the user that give the highest similarity value in the attack. However, if another query has a similar or slightly lower similarity, the user and the query returned by the attack might not be relevant as the two queries would have approximately the same probability to be the original one.

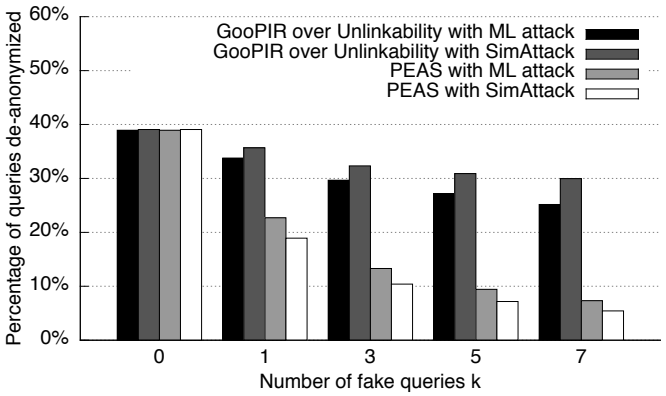
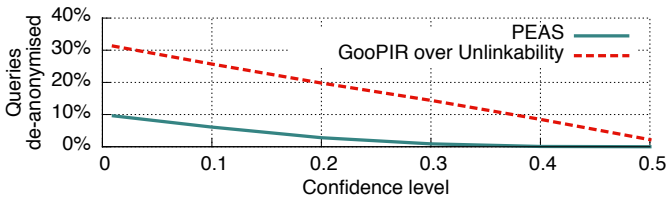
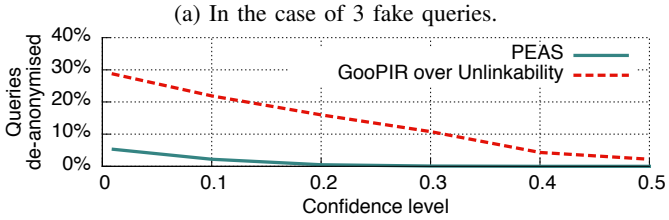


Fig. 3. Comparison between PEAS obfuscation and GooPIR obfuscation over an unlinkability solution depending on the number of fake queries.



(a) In the case of 3 fake queries.



(b) In the case of 7 fake queries.

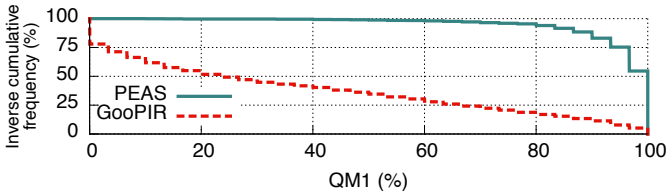
Fig. 4. Confidence level of the adversary using SimAttack.

That is why, we define the confidence level as the difference between the scores of two first results of the attack. The value of this metric ranges from 0 to 1. Figure 4 shows the results of SimAttack (ML attack gives similar results) applied on queries obfuscated with 3 and 7 fake queries with various confidence levels for both PEAS and GooPIR. Results show that if the adversary wants a confidence level of 0.2 the percentage of de-anonymized queries drops respectively to 2.8% and 0.5% with 3 and 7 fake queries. In the same situation, GooPIR still fails to protect 19.8% and 16% of queries.

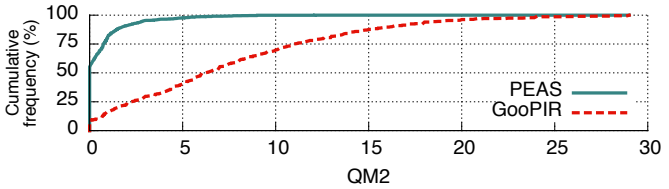
C. Accuracy

In this section we evaluate the accuracy of PEAS by showing the impact of the obfuscation and filtering techniques on the information retrieval results. Practically, we study if the filtering technique is able to remove irrelevant results (i.e., those relating to fake queries) while keeping the relevant ones (i.e., those relating to the initial query).

1) Methodology and Metrics: We used Google to obtain the non-personalized results of the queries. As the OR operator implemented by Google only works with single-word queries, we simulated the execution of an obfuscated query $q^+ = q_0$ OR ... OR q_k by submitting each sub-query q_i independently and by merging the $(k+1)$ result sets. Besides,



(a) Percentage of expected results retrieved after filtering.



(b) Comparison between the rankings of the filtered results and the expected ones.

Fig. 5. Accuracy of the results returned by PEAS architecture filtered by PEAS and GooPIR filterings.

due to the query limitations of Google, we run the evaluation on a subset composed of 1,000 queries. To assess the quality of the results obtained by our system, we use the two same metrics as in the GooPIR paper. The first metric QM_1 is the percentage of initial results retrieved by the system:

$$QM_1 = \frac{card(R \cap \bar{R})}{card(R)} \times 100$$

where R and $\bar{R}$ are the lists of results returned respectively by Google and PEAS. The results returned by Google is, in this experiment, the baseline while the results returned by PEAS is the set of results we are studying. QM_1 equals 100 when R and $\bar{R}$ contain the same elements.

The second metric QM_2 is the average, over the results that are both in R and $\bar{R}$, of the absolute rank differences. It is defined as:

$$QM_2 = \frac{\sum_{r \in R \cap \bar{R}} |rank_R(r) - rank_{\bar{R}}(r)|}{card(R \cap \bar{R})}$$

where $rank_L(e)$ denotes the rank of an element e in a list L. This metric allows to compare the rankings of the expected list of results R with another list of results $\bar{R}$. QM_2 equals 0 if and only if the elements of $(R \cap \bar{R})$ appear at the same position in R and $\bar{R}$. In this evaluation we consider top-30 queries, i.e., for each query we consider the first 30 results.

2) Evaluation of the Accuracy: Figure 5(a) shows the inverse cumulative frequency distribution of the metric QM_1. It shows that in more than 50% of cases, PEAS returns all the expected results (i.e., the results returned when no obfuscation technique is applied). In 95% of cases, PEAS returns more than 80% of the expected results, that is 24 correct results out of 30.

Figure 5(b) presents the cumulative frequency distribution of the metric QM_2. It shows that the rankings of the results returned are identical to the expected results in more than 50% of cases. Moreover, in 90% of cases, we find that $QM_2 \leq 2$, which means that the rank of de-obfuscated results differ on average with the expected results by 2 units.

TABLE II
NUMBER OF CRYPTOGRAPHIC OPERATIONS AND TRAFFIC GENERATED IN THE WHOLE SYSTEM.

Protocol	Encryption		Decryption		Traffic
	RSA	AES	RSA	AES	
Direct access	0	0	0	0	5
PEAS	1	1	1	1	15
Onion Routing	2	6	2	6	39

From these two experiments, we observe that PEAS filtering retrieves significantly more results than GooPIR (on average 95.3% for PEAS and 36.3% for GooPIR). Besides, the results returned by PEAS are more accurate than GooPIR as the ranking of results (compared to the expected results) differ on average by 0.6 unit for PEAS compared to 5.8 units for GooPIR.

The results presented on Figures 5(a) and 5(b) demonstrate that PEAS preserves a good accuracy of the results as both the retrieved results and their ordering are marginally impacted by the obfuscation technique.

D. Performance

In this section, we evaluate the performance of our privacy proxy. We compare it with the Onion Routing protocol, as it is, to the best of our knowledge, the most efficient anonymous communication protocol. In order to have a fair comparison of both protocols, we implemented and configured the Onion Routing as follow: (*i*) we set up only two relays (as our privacy proxy is composed of two servers and using more relays would produce a higher overhead for Onion Routing), and (*ii*) we configured the protocol to re-negotiate the symmetric keys for each query (as it prevents the exit node from using the key to link queries coming from the same user). Finally, we assume that clients and servers of both protocols know all the necessary public keys in advance.

1) Theoretical Evaluation: We distinguish three metrics: the number of symmetric encryption and decryption operations performed, the number of asymmetric encryption and decryption operations performed, and the traffic generated. We compute these metrics for the whole system. Table II shows the values of these metrics when using PEAS, Onion Routing and when directly accessing the search engine. These results demonstrate that PEAS requires less cryptographic operations than Onion Routing (twice as less RSA encryption/decryption and six times less AES encryption/decryption operations). In terms of network traffic, PEAS sends 38% less messages than Onion Routing. Indeed, querying a website using PEAS requires to send 15 messages (9 for the TCP handshake and 6 for the protocol messages), while using Onion Routing requires to send 39 messages (9 for the TCP handshake, 18 for the TLS handshake and 12 for the protocol messages).

2) Round Trip Time: We measure the time delay on the client side, from the submission of a query to the deciphering of the received response. To avoid uncertainty in the Internet network latency and in the search engine query processing, we simulate a search engine response on the last node (i.e., issuer

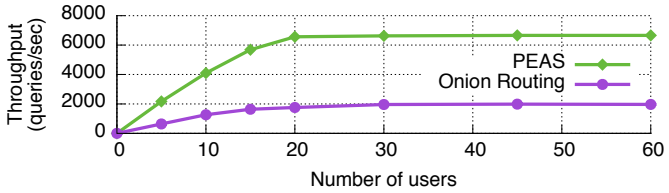


Fig. 6. Comparison between the scalability of PEAS and Onion Routing.

or exit node). We consider one client using the architecture. We execute the PEAS implementation and the Onion Routing implementation on three identical servers (one as a client and two as a server). The average PEAS round trip time (measured on 50,000 queries) is 5.97 times lower than the Onion Routing protocol (2.761ms for PEAS and 16.475ms for Onion Routing). This difference is mainly due to the number of cryptographic operations (as shown in Table II) and to the re-negotiation of the symmetric keys for each query performed by the Onion Routing protocol.

3) Scalability: We evaluate the scalability of our architecture compared to Onion Routing. Figure 6 shows the throughput as a function of the number of clients using the two architectures. It shows that both architectures handle approximately the same number of users (i.e., 20) before saturation of the throughput. Nevertheless, PEAS architecture has a throughput three times higher than Onion Routing (i.e., 6,695 queries/sec for PEAS and 2,034 queries/sec for Onion Routing). As described in Section IV-E, a higher scalability might be reached with a distributed architecture composed of multiple receivers and issuers.

VII. Conclusion

This paper presents PEAS, a new protocol enabling users to query search engines in a more privacy-preserving way. PEAS is composed of two techniques: the first one hides the identity of the user while the second obfuscates user's queries and then filters out the results to remove the noise introduced by the obfuscation mechanism. We demonstrate theoretically that the first techniques guarantees unlinkability between users and the queries they issue. We conducted experiments with real data to validate the whole approach. Results demonstrate that PEAS ensures a high privacy protection: less queries are de-anonymized using PEAS than using Tor, TrackMeNot or GooPIR. Besides, PEAS is efficient (it performs less cryptographic operations than Tor) and accurate (results retrieved using PEAS are similar to those obtained without obfuscation).

As future work, we plan to take into account the sensitivity of the queries as discussed in [24]. Hence, instead of obfuscating all the queries, we will aim at adapting the obfuscation mechanism to target sensitive queries. The objective is to improve the performance while preserving the same level of privacy.

Acknowledgment

The presented work was supported the EEXCESS project funded by the EU Seventh Framework Program, grant agreement number 600601.

References

[1] D. Goldschlag, M. Reed, and P. Syverson, "Onion routing," *Communications of the ACM*, vol. 42, no. 2, 1999.
[2] R. Dingledine, N. Mathewson, and P. Syverson, "TOR: The second generation onion router," in *Usenix Security Symposium*, 2004.
[3] D. I. Wolinsky, H. Corrigan-Gibbs, and B. Ford, "Dissent in numbers: Making strong anonymity scale," in *Proceedings of OSDI*, 2012.
[4] S. Ben Mokhtar, G. Berthou, A. Diarra, V. Quéma, and A. Shoker, "Rac: A freerider-resilient, scalable, anonymous communication protocol," in *Proceedings of ICDCS*, 2013.
[5] S. T. Peddinti and N. Saxena, "On the effectiveness of anonymizing networks for web search privacy," in *Proceedings of the 6th ACM ASIACCS*. ACM, 2011, pp. 483–489.
[6] V. Toubiana, L. Subramanian, and H. Nissenbaum, "Trackmenot: Enhancing the privacy of web search," *arXiv preprint arXiv:1109.4677*, 2011.
[7] J. Domingo-Ferrer, A. Solanas, and J. Castellà-Roca, "h(k)-private information retrieval from privacy-uncooperative queryable databases," *Online Information Review*, vol. 33, no. 4, pp. 720–744, 2009.
[8] M. Murugesan and C. Clifton, *Providing Privacy through Plausibly Deniable Search.*, 2009, ch. 65, pp. 768–779. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9781611972795.66
[9] H. Pang, X. Ding, and X. Xiao, "Embellishing text search queries to protect user privacy," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 598–607, 2010.
[10] B. Blanchet, "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules," in *Proceedings of the Computer Security Foundations Workshop (CSFW)*, 2001.
[11] G. Pass, A. Chowdhury, and C. Torgeson, "A picture of search," in *Proceedings of the 1st International Conference on Scalable Information Systems*, ser. InfoScale '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1146847.1146848
[12] M. Shapiro, "Structure and Encapsulation in Distributed Systems: the Proxy Principle," in *Proceedings of ICDCS*, 1986.
[13] J. Castellà-Roca, A. Viejo, and J. Herrera-Joancomartí, "Preserving user's privacy in web search engines," *Computer Communications*, vol. 32, no. 13, 2009.
[14] Y. Lindell and E. Waisbard, "Private web search with malicious adversaries," in *Proceedings of PETS*. Springer, 2010, pp. 220–235.
[15] R. Al-Rfou, W. Jannen, and N. Patwardhan, "Trackmenot-so-good-after-all," *arXiv preprint arXiv:1211.0320*, 2012.
[16] D. Rebollo-Monedero and J. Forné, "Optimized query forgery for private information retrieval," *Information Theory, IEEE Transactions on*, vol. 56, no. 9, pp. 4631–4642, 2010.
[17] S. Ye, F. Wu, R. Pandey, and H. Chen, "Noise injection for search privacy protection," in *Computational Science and Engineering, 2009. CSE'09. International Conference on*, vol. 3. IEEE, 2009, pp. 1–8.
[18] A. Arampatzis, P. S. Efraimidis, and G. Drosatos, "A query scrambler for search privacy on the internet," *Information retrieval*, vol. 16, no. 6, pp. 657–679, 2013.
[19] M. Juarez and V. Torra, "Dispa: An intelligent agent for private web search," in *Advanced Research in Data Privacy*. Springer International Publishing, 2015, vol. 567, pp. 389–405.
[20] M. Barbaro, T. Zeller, and S. Hansell, "A face is exposed for aol searcher no. 4417749," *New York Times*, vol. 9, no. 2008, p. 8For, 2006.
[21] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.
[22] Google Inc., "Google trends," 2012. [Online]. Available: http://www.google.com/trends/
[23] S. Guha, M. Jain, and V. N. Padmanabhan, "Koi: A location-privacy platform for smartphone apps." in *NSDI*, 2012, pp. 183–196.
[24] A. Petit, S. Ben Mokhtar, L. Brunie, and H. Kosch, "Towards efficient and accurate privacy preserving web search," in *In Proc. of the 9th Workshop on Middleware for Next Generation Internet*, 2014.