CrowdLogging: Distributed, Private, and Anonymous Search Logging

Henry Feild
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
hfeild@cs.umass.edu

James Allan
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
allan@cs.umass.edu

Joshua Glatt
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
jglatt@cs.umass.edu

ABSTRACT

We describe CrowdLogging, an approach for distributed search log collection, storage, and mining, with the dual goals of preserving privacy and making the mined information broadly available. Most search log mining approaches and most privacy enhancing schemes have focused on centralized search logs and methods for disseminating them to third parties. In our approach, a user's search log is encrypted and shared in such a way that (a) the source of a search behavior artifact, such as a query, is unknown and (b) extremely rare artifacts—that is, artifacts more likely to contain private information—are not revealed. The approach works with any search behavior artifact that can be extracted from a search log, including queries, query reformulations, and query-click pairs. In this work, we: (1) present a distributed search log collection, storage, and mining framework; (2) compare several privacy policies, including differential privacy, showing the trade-offs between strong guarantees and the utility of the released data; (3) demonstrate the impact of our approach using two existing research query logs; and (4) describe a pilot study for which we implemented a version of the framework.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*data mining*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*search process*; H.3.4 [**Information Storage and Retrieval**]: Systems and software—*distributed systems, user profiles and alert services*

General Terms

Algorithms, Design, Experimentation, Management, Security

Keywords

Distributed query logs, private query log analysis

SIGIR'11, July 24–28, 2011, Beijing, China.

1. INTRODUCTION

Query logs have been a windfall for search engine companies, providing better ranking [15], improved spelling correction [2], more useful query suggestions [5], and generally helping researchers understand how people search [13]. In its simplest form, a query log is the list of queries received by a search engine, along with details of which results the user viewed or clicked. Search providers often enhance the logged information to the extent possible, including asking their users to install "toolbars" on their Web browsers that capture browsing activity across different search providers and all other websites.[1]

With a query log in hand, it is possible to puzzle out not only a great deal about the community of users, but also about individuals: the queries issued and URLs of web pages viewed by each user are connected through an approximate user ID, a browser cookie. It is easy to pull together all the data for a single person, no matter how private or sensitive the information is. The well-known AOL query log incident demonstrated how even anonymizing usernames with random identifiers is not adequate in protecting the identity of individuals: several users were identified by a *New York Times* reporter [4]. User 4417749 was identified through a combination of several infrequent queries conveying private information, such as zip codes and relatives' names. Once that individual was identified, all of her sensitive queries were exposed, in this case providing a disturbingly clear picture of her interests and activities.

The enormous value of query logs, however, means that despite the privacy concerns, many people and organizations are constantly interested in acquiring them. These logs are readily available at large search companies [19, 24, 25], but smaller groups—including academic researchers—have few ways of acquiring such logs. Some logs have been released to the general research community, often after careful efforts to remove all personal information.[2] Unfortunately, not only are the resulting logs artificial (because data is lost), but they are infrequent and short-term snapshots that are adequate only for limited types of research.

To address those issues, we present an approach to capturing and using query logs that (1) provides log access to anyone and (2) provides privacy guarantees that are equal or superior to all existing methods.

[1] http://www.discoverbing.com, http://toolbar.google.com, http://toolbar.yahoo.com
[2] http://ims.dei.unipd.it/websites/LogCLEF/Overview.html

All current search log mining and anonymization models we know of are based on a centralized approach. The entire search log is collected and stored by a single entity, such as a search engine company. This collector can mine the log for its own purposes, of course—and, if they choose, provide some privacy mechanism by which researchers can analyze the data. For example, this privacy mechanism might be query-based, where the researcher poses questions to the collector ("how often has *unusual search* been issued as a query?"), or distribution-based, where the collector creates an anonymized version of the search log, which is then given directly to the researcher.

These centralized approaches work well for collectors that collect their data through a Web service, such as a search engine company. But the drawbacks are considerable: (1) users have virtually no control over their query or click data once it has been collected; (2) shared versions of log data are usually delayed, censored, down-sampled, or modified, thus reducing their utility; and (3) users' privacy is at the mercy of the collector's security capabilities.

We present CrowdLogging, an alternate approach that keeps each user's log on their own machine.[3] In this model, mining tasks are partially handled at each user's site; the results are then relayed to one or more central servers that complete the process. The results of the mining are encrypted in such a way that they can only be decrypted if the data is sufficiently common across many users, making it nearly impossible for a user's private information to be revealed: we will never see the query *ID number 999-99-9999*, because only one person will ever type it in. In addition, the source of log information is stripped, providing a degree of anonymity not normally available—one that means a common but embarrassing query is not connected to any individual: we know that *embarrassing query* was issued by many people, but we do not know who they are.

The combination of these two privacy features yields a data mining system based on distributed, private, and anonymous search logs. There are several advantages of this approach: (1) users have a high degree of control over their collected data, including being able to opt out selectively or entirely; (2) the original, un-anonymized data on each user's computer can be used immediately or with a delay for as many mining tasks as the the user and the security policy allow, and (3) the information collected by the server is either common across many users and disassociated from them all, or is encrypted in such a way that it cannot be determined by someone illegitimately gaining access to the server—or even by someone who has legitimate access to the server, such as by subpoena.

Our contributions include the following: (1) a distributed search log collection, storage, and mining framework, presented in Section 3.1; (2) a comparison of four privacy policies, focusing on the trade-offs between the strength of the guarantees on privacy loss and the utility of the released data, covered in Section 3.2; (3) simulations showing the impact of the framework, using two existing research query logs, described in Section 4; and (4) a pilot study for which we implemented a version of the framework, discussed in Section 5.

[3]It is not necessary that the logs be kept on the user's computer, though doing so maximizes the user's control and the flexibility of the mining tasks and we will focus on that approach in this study.

2. RELATED WORK

The motivation of our work is similar to that of most work on search log anonymization: how can search log information be collected, used, and possibly shared so that the users who are logged will not have private or embarrassing information revealed? Most work in this area has focused on ways to transform a centrally located search log into one that does not violate privacy concerns.

Adar [1] presents two methods for anonymizing a search log in the interest of preserving privacy. First, only queries that are issued by at least t users are kept. Adar uses a secret sharing scheme where queries are encrypted and can only be decrypted if a sufficient number of parts are received. This is similar to Sweeney's k-anonymity work [23] for databases, which assigns a subset of the database's columns to be a quasi-identifier. A row in the database is only released if for each column value in the row's quasi-identifier, there are at least $k-1$ other rows that share the column value.[4] Under Adar's model, a query serves as the quasi-identifier. Adar's second technique anonymizes the identity of users who enter a series of queries that may reveal who they are when taken together, by clustering syntactically related queries from a user's session and releasing each cluster under its own identifier. Our approach uses a similar form of t-anonymity but does not cluster queries, because we do not link a user's various queries: even though individual queries are t-anonymous, there is no guarantee that the syntactic clusters do not contain both identifying and sensitive information when all the queries in the cluster are looked at together. Clustering also rearranges the query log, creating false associations between queries and discarding potentially interesting sequences of queries (e.g., reformulations).

Hong et al. [11] also present a technique for anonymizing search logs. They define k^{δ}-anonymity, where for every user whose data is listed in the anonymized query log, there are $k-1$ other users that are δ-similar to the user in terms of their data. Their method involves grouping similar users and adding or suppressing data to make user groups appear more similar. The downside to this approach is that some data is permanently discarded and artificial data is added. It is unclear how different types of applications are affected by this anonymization process.

Korolova et al. [16] describe a differentially private method for releasing a query click graph that protects privacy. Their work differs from others in this field in two important ways: (1) the output of their anonymization is a query click graph rather than a search log, and (2) they prove theoretical bounds on the effectiveness of their privacy-protecting scheme, under the assumption that an attacker has an arbitrary amount of outside information. Since *differential privacy* [7] is used, a large amount of data is lost and noise is added, a common trade-off for theoretically sound privacy guarantees. In Section 3.2.3, we show how a portion of their method can be adapted for use as a privacy policy in our framework.

The common thread through these works is that data is permanently lost in the anonymization process, but the

[4]There is extensive work on this type of anonymization in databases, including methods that do not require anonymized sending and that are robust in the presence of malicious participants [26]. Some of those ideas may be applicable in our setting, but our situation is simpler—we have only one "column"—and allows for a simpler apparatus.

loss may be a necessary evil to distribute search logs—particularly to convince legal counsel that the data is sufficiently sanitized. An alternative approach for universities or small organizations providing search is to gather query log information from volunteers. The Community Query Log Project[5] at the Center for Intelligent Information Retrieval was one such attempt to do that. The project distributed a browser toolbar that collected search log information, anonymized the source, and shipped it to a central server. The intent was to review the data to ensure it was sufficiently private and then release it to the research community. Unfortunately, after a year of effort only 34,000 distinct queries were collected—according to the project, the equivalent of six seconds of Google traffic. The project was abandoned in May 2010. A conjecture that motivates our work is that user privacy concerns were a major reasons for poor adoption: the idea was appealing, but people were nervous about participating.

Another body of work attempts to address privacy concerns differently. The TrackMeNot project [12], for example, inserts random queries into the stream of queries issued by a user, with the intent of making it harder for a search engine company to determine a particular user's interests. This provides a degree of privacy, but it makes search logs less useful by inserting additional noise that makes a user's general interests difficult to discern. However, personal or embarrassing queries are still present in the log and TrackMeNot thus does nothing toward making them distributable.

An entirely different approach to making query log information revolves around policy. For example, Bar-Ilan [3] argues for setting up review boards that would allow researchers to use logs under controlled and clearly defined conditions. It is not clear that such a review board would ease privacy concerns sufficiently. Moreover, the limitations such an approach demands would likely disincentivize most researchers.

Our belief is that all of the previous work is problematic in that it rests on the assumption that query logs should be collected in the clear and processed by groups or companies that we trust to store, use, and share the information with our best interests at heart. However, no matter how good their intentions, they may be thwarted by hackers who steal the information [21] or governments that demand access [10].

3. CROWDLOGGING

Our framework, CrowdLogging, consists of three main layers: the system, the privacy policies, and the applications. The system and applications are shown in Figure 1. The privacy policy layer is not explicitly depicted in the figure; it provides constraints on how results are mined, encrypted, aggregated, and released. We describe each of the layers below.

3.1 System

In current search log collection and mining systems (Figure 2), raw browser behavior data is sent from users (by their browsers) to a centralized collector where it is stored. When analysts want to process the data, the collector mines the raw data and then selectively chooses which bits to reveal or suppress in order to respond to the queries posed by analysts. Individual users have no control over how

[5]http://lemurstudy.cs.umass.edu/

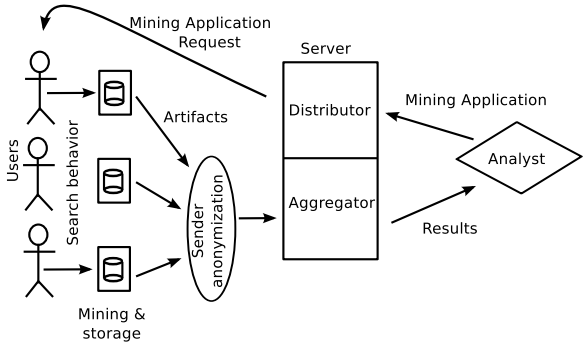

Figure 1: The distributed, private, and anonymous search log mining system.

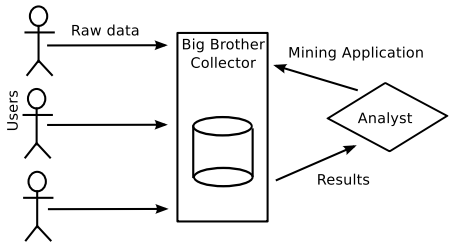

Figure 2: A typical centralized collection, storage, and mining system.

their raw data—which may contain sensitive and identifying information—is used in these mining tasks, and the collector has the ability to look through the raw data at will. Our system moves away from this centralized system model and provides mechanisms for giving users control of their data, keeping their data private, and anonymizing their data.

Distributed. We distribute the storage of raw data to the users that generate the data. A local Web proxy or Web browser extension stores a user's browsing behavior—such as queries submitted to search engines and Web pages visited—to a log located *on that user's computer*. This gives users full control over their raw information—they can remove the log, remove parts of the log, and choose in which mining tasks to participate, depending on the control mechanisms implemented.

Private. In the centralized approach, raw data, such as a query submitted by the user to a search system, is communicated to the collector. In our system, the raw data stays on the users' computers. Only the results of a mining task, which we call *search artifacts*, can leave a user's computer.

To provide privacy, the artifacts are encrypted using a "secret sharing" scheme. Such schemes encrypt an artifact and send with it a piece of the key necessary for decryption. If a sufficient number of distinct key pieces are collected for a particular encrypted artifact, it can be decrypted.

There are different ways to pick which part of the key to include with an encrypted artifact, some of which we will discuss in Section 3.2. In general, the function that generates the key part has four parameters: the artifact, the user's ID, k, and n. k is a number that dictates how many distinct key parts are needed for decryption and n is the number of distinct key parts that are available; $k \leq n$.

A key part generation function that serves as a good example produces different key parts for different combinations of the artifact, user ID, k, and n. This allows an artifact to

be decrypted if at least k users generated the same artifact, a specialized version of k-anonymity.

Anonymous. Users' anonymity is protected in two major ways within our system. First, encrypted artifacts are required to go through an anonymizing system before arriving at the aggregation server. This prevents the aggregation server from having the capability of associating potentially identifying information, such as an IP address, with an artifact.

Second, the aggregator provides additional anonymity by not releasing key part information. Depending on how the key part is determined, artifacts from a single user may be linked by part number. Although this connection with the user can be removed, we implement a policy of dropping the part numbers in an abundance of caution. This policy ensures that no information allowing two artifacts to be linked to a single user is released, maintaining users' anonymity.

As with any system, there are security vulnerabilities. We outline these in Section 5.4.

3.2 Privacy policies

In this section, we will describe a sample of privacy policies that are compatible with CrowdLogging. A privacy policy answers four questions: (1) what data can be mined and sent to the server, (2) how it is packaged for sending to the server, (3) how data is aggregated at the server, and (4) what is released to analysts.

3.2.1 Artifact frequency thresholding

We define the artifact frequency thresholding privacy policy (FT_a) to mean that an analyst can access any search artifact mined from the logs of users provided it occurs at least k times. *What can be mined?* There are no constraints on what gets mined from the log. *How does it get packaged?* Each artifact is encrypted and a random key part is assigned. If a particular artifact occurs more than once for a single user, each instance is encrypted with a different key part, though there will be some overlap due to the random generation. *How is data aggregated?* The keys for identical encrypted artifacts are combined; if there are at least k of them, the artifact is decrypted. *What is released?* Pairs of artifacts and counts. If the artifact was decrypted during aggregation, the decrypted form is released. Otherwise, a meaningless string is substituted.

This is a weak privacy policy. A user could enter a query that is both identifying and sensitive k times, causing it to be revealed. Potentially sensitive non-query artifacts could also be extracted. Regardless, the framework allows this policy, and it can be used in conjunction with more private policies, such as artifact frequency differential privacy.

3.2.2 User frequency thresholding

The user frequency thresholding policy (FT_u) is based on k-anonymity as presented by Sweeney [23]. The policy states that an artifact can only be revealed if it is entered by at least k distinct users. *What can be mined?* There are no constraints on what can be mined. *How does it get packaged?* Each artifact must be encrypted and the key part must be such that the same user has the same key part for all instances of the same artifact. *How is data aggregated?* Data is aggregated the same way as under the FT_a policy. *What is released?* The same information is released as under the FT_a policy.

The motivation for FT_u in the context of this work is that an artifact that is extracted from at least k distinct users' logs is not likely to contain sensitive information. For example, consider a particularly sensitive query that contains an individual's name and ID number. It is conceivable that the query will be released under the FT_a policy for a low threshold of k. However, even for a low value of k, it is unlikely that $k-1$ other users will have entered that same sensitive query.

3.2.3 Differential privacy

A drawback of the policies above is that they do not provide any provable guarantee of privacy. A method that has been proposed to quantify privacy loss is differential privacy [7]. The motivation is that, given a database and a query, we want to produce a noisy answer such that the querier has a low probability of determining if any individual was included in the database.

Korolova et al. [16] present a differentially private algorithm for producing a query click graph from a search log. Their algorithm has four steps: (1) take the first d queries from each user; (2) for each query, add some noise to its frequency in the data set and select it for release if it surpasses a threshold K; (3) for released queries, add noise to their frequencies; and (4) release noisy counts for clicks on URLs shown in the top ten results for each query.

There are three major constraints that differential privacy puts on search log mining in order to make the desired guarantees: the number of artifacts mined from a particular user must be limited; Laplace noise must be added to any artifact counts before comparing with the threshold; and each mining job depletes some amount of the privacy quota for a data set, so only one or a small number of mining tasks can be performed on a data set, depending on the parameters.

We present two variations on this algorithm and the formulas behind them to provide additional privacy policies for our framework. In both variations, because our goals are more general, we replace queries with arbitrary search artifacts and we omit the final step of releasing URL click counts.

Artifact frequency differential privacy (DP_a). This algorithm is similar to Korolova et al. [16]: (1) use the first d artifacts from each user; (2) for each artifact, add some noise to its frequency in the data set and select it for release if it surpasses a threshold k_a; (3) for released artifacts, add noise to their frequencies. This provides $(d\ln(\alpha) + d/b_a, \delta)$-differential privacy, where $\alpha = \max\left(\exp(1/b), 1 + \frac{1}{2\exp((k_a-1)/b)-1}\right)$, b is the Laplace noise parameter for step (2), b_a is the Laplace noise parameter for step (3), and $\delta = \frac{d\exp((d-k_a)/b)}{2}$. Because a search artifact can be viewed as a string, just as a query is, the proof for this variation follows directly from the proof presented by Korolova et al. [16]. However, they use K instead of k_a and b_q in place of b_a.[6]

Assuming, as Korolova et al. do, that we want to minimize noise and that $\exp(1/b) \geq 1 + \left(\frac{1}{2\exp((k_a-1)/b)-1}\right)$, we can maximize the threshold k_a by solving $k_a = d\left(1 - \frac{\ln(2\delta/d)}{\epsilon}\right)$, where ϵ is the privacy quota. The implementer needs to

[6]We use this alternative naming to easily differentiate between the two variations we present.

select values for ϵ, δ (less than the inverse of the number of users participating), and d.

What can be mined? Anything can be mined, but only d artifacts can be sent to the server. *How does it get packaged?* Follow the same scheme as for FT_a, except that in addition, an encrypted version of the user ID must be included for the next step. *How is data aggregated?* First, ensure that only d artifacts are present for each user; as a single user may have multiple machines, and the machines do not communicate, it is possible for there to be multiple d-artifact sets sent to the server for the same user ID. Once there are d artifacts per user, data is aggregated the same way as under the FT_a and FT_u policies. *What is released?* The same information is released as under the FT_a and FT_u policies.

User frequency differential privacy (DP_u). This variation differs in step (2) of DP_a: for each artifact, add some noise to *the number of distinct users with that artifact* and release it if that number is at least k_u. Due to the nature of this change, we can modify the proof slightly to obtain a tighter bound on privacy. Specifically, looking at Equation 9 in Korolova et al. [16], we have

$$\frac{1}{2}\sum_{i=1}^{n_y}\exp\left(\frac{M(y_i,D_2)-k_a}{b}\right)\leq\ \frac{d}{2}\exp\left(\frac{d-k_a}{b}\right) \qquad (1)$$

where D_2 is a dataset with a single user added, versus D_1, which is the same dataset but with that user removed, $y_1,\ldots,y_{n_y}$ is the set of queries which are unique to D_2, and $M(y_i,D_2)$ returns the frequency of the artifact y_i in D_2. $M(y_i,D_2)$ can return at most d, since a user can only contribute at most d artifacts. However, under the user-frequency model, let $M'(x,D)$ return the number of distinct users associated with an artifact x in D. Thus, $M'(y_i,D_2)$ is at most 1, since y_i is unique to the one user whose data is in D_2 but not D_1. This yields the following, tighter bound:

$$\frac{1}{2}\sum_{i=1}^{n_y}\exp\left(\frac{M'(y_i,D_2)-k_u}{b}\right)\leq\ \frac{d}{2}\exp\left(\frac{1-k_u}{b}\right) \qquad (2)$$

This has ramifications for δ, giving us the following optimal user-frequency threshold:

$$k_u = 1-\frac{d\ln(2\delta/d)}{\epsilon} \qquad (3)$$

What can be mined? Anything can be mined, but only d artifacts can be sent to the server. *How does it get packaged?* Follow the same scheme as for FT_u, except that in addition, an encrypted version of the user ID must be included for the next step. *How is data aggregated?* Follow the same procedure as for DP_a. *What is released?* The same information is released as under the other three policies.

3.3 Applications

CrowdLogging allows any mining task—i.e., applications—to be distributed to consenting clients. An application starts with a user's local search log and produces search artifact instances as output, possibly limited by the selected privacy policy. In this section we describe several popular mining applications and show how to implement them in the CrowdLogging framework. We discuss counting queries, query reformulations, query-URL pairs, and learning-to-rank features.

Query frequency mining. We wish to emit all occurrences of queries as search artifacts. Since a query can be submitted to a search engine in many different forms that are semantically identical—e.g., with different spacing or capitalization—we use both the original query and a normalized form that collapses those minor differences. The client therefore sends the server two encrypted pieces of data: the normalized query as the *primary private field* and the original query as the *secondary private field*. The two fields are encrypted with the same key which is derived from the primary private field—that is, the secondary private field will be decryptable only once the primary private field is decryptable, regardless of how many times that particular unnormalized form of the query occurs. The format for the message sent to the server is thus:

$$\langle E(N(query_1)), E(query_1), key\text{-}portion\rangle$$

where $N(x)$ is a function that returns the normalized version of the string x. When the encrypted artifacts arrive at the server, the server aggregates the tuples based on the primary private key. Once there are a sufficient number of distinct key portions, both private fields can be decrypted. We can then generate and distribute to analysts tuples like:

⟨ "dog food", "DOG FOOD", 1 ⟩
⟨ "dog food", "dog food", 100 ⟩

This is enough information to tell us how frequently a normalized query occurred in the logs as well as the frequency of variations of the query. Depending on the privacy policy, we can aggregate over undecrypted tuples and generate aggregate statistics such as: "5 undecrypted queries were each entered 194 times; 20 undecrypted queries were each entered 195 times." These statistics allow analysts to look at potentially helpful statistics about rare items in a collection without revealing the queries themselves.

Query reformulation mining. The next application we consider is mining query reformulations. Query reformulations occur when a user issues a query and then shortly thereafter tries a different one. Although in many cases the queries are completely unrelated, there are numerous times when they are connected. For example, Jansen et al. report that over half of the users in a 24-hour query log modified their queries [14]. Query reformulations can represent spelling corrections, related words, acronym expansions, or alternate ways of asking the same question.

To find common query reformulations, we can replace the single query in the previous application with a pair of queries, using a tab or other delimiter to separate them—we use a colon in the examples below. The pairs could be restricted to those that are adjacent or to all pairs within a several-minute session. Depending the exact specifications of the application, the pairs might be time-ordered or not. What we send is:

$$\langle E(N(q_{1,1}:q_{1,2})), E(q_{1,1}:q_{1,2}), key\text{-}portion\rangle$$
$$\langle E(N(q_{2,1}:q_{2,2})), E(q_{2,1}:q_{2,2}), key\text{-}portion\rangle$$

As in the case of counting queries, the server can count query reformulations and the ones with a sufficient number of distinct key portions can be revealed. If the privacy policy allows for it, this application can be run in tandem with query frequency mining. This allows the server to tabulate how frequent a query is and what percent of its instances are reformulated in a particular way. With this information, the server can derive useful statistics, such as 'the query "dog"

is rewritten as "puppy" 50% of the time.' It can also report the most frequent and widely supported reformulations and the proportion of all reformulations they make up.

Compared to non-private query logging, what have we lost? Certain reformulation pairs will be infrequent and so will not be discoverable. For example, we may be able to say that,

"dog"	→	"puppy"	: 50%
"dog"	→	"dog food"	: 20%
"dog"	→	?	: 30%

We cannot know how many additional reformulations are encompassed by the unknown 30%, but we do know the proportion of reformulations for "dog" that we are missing.

Query-URL frequency mining. Another activity that is useful to mine is which pages are clicked on in response to a query and, the flip side, which queries are used frequently to get to a page. Both provide information related to relevance: pages that are probably relevant to the query and words (from queries) that are strongly related to a page.

To discover this information, we use the same approach as for finding query reformulations, except that we provide a query-page pair rather than a query-query pair. The mining mechanism sends the following encrypted search artifact:

$$\langle E(N(q_1 : URL_1)), E(q_1 : URL_1), \textit{key-portion}\rangle$$
$$\langle E(N(q_1 : URL_2)), E(q_1 : URL_2), \textit{key-portion}\rangle$$

where q_i is some query and URL_j is a page (URL) that the user subsequently clicked on, presumably in response to the query. As before, the server does not know who issued the queries that resulted in the clicks. A way to increase the commonality is to reduce the URL to just the domain or a base-level directory.

An obvious variation is to drop the query q_i from the tuple and send only information about pages that are clicked, providing *a priori* measures of popularity for web pages. This would look very similar to the query frequency mining task, but instead of a query, we use a URL.

Learning to rank feature mining. A common setup for a learning to rank (LTR) system is to specify a list of page feature vectors along with a relevance level for an associated query. An LTR system aims to learn how the page features are related to the relevance levels and will place pages with higher relevance levels higher in the ranked list.

One way to support LTR in our approach is as follows. Concatenate a query, a clicked URL, and the list of features as the search artifact. This will yield something like this:

$$\langle E(N(q_1 : URL_1 : f_1 : f_2 \ldots)), E(\ldots), \textit{key-portion}\rangle$$
$$\langle E(N(q_1 : URL_2 : f_1 : f_2 \ldots)), E(\ldots), \textit{key-portion}\rangle$$

where q_i is a query, URL_i is a URL clicked on for a query, and f_i is a feature for a query-URL pair. A pitfall of this approach is that the full list of features are unlikely to be common taken together.

A slightly more robust approach breaks the features into pieces, as follows:

$$\langle E(N(q_1 : URL_1 : \text{F1:}f_1)), E(q_1 : URL_1 : \text{F1:}f_1), \textit{key-portion}\rangle$$
$$\langle E(N(q_1 : URL_1 : \text{F2:}f_2)), E(q_1 : URL_1 : \text{F2:}f_2), \textit{key-portion}\rangle$$

Each search artifact consists of a query q_i, a URL URL_i, and one of the features, f_i.

To improve the commonality, the URL could be shortened, as mentioned above. In addition, the features could be mapped into buckets.

4. SIMULATIONS

One of the motivations for query log privacy comes out of the embarrassment caused by AOL's release of query logs. That leads us to explore the impact our approach would have on the users in that log, widely available on the Internet.[7] The log covers three months of activity, from March through May of 2006. It includes more than 10 million unique queries making up 21 million query instances and covers more than 650,000 unique users.

In addition, we analyze the effectiveness of our approach on a 2006 MSN search log. The MSN search logs use session IDs rather than user IDs. This is not ideal as two sessions could originate from a single user, but it offers a different perspective from the AOL log. The log contains a sample of 15 million query instances submitted to the MSN search engine in May 2006, 13 million non-duplicate adjacent queries, 6.6 million distinct queries normalized in casing and spacing, and 7.5 million sessions.

We want to understand how much data would be lost and how well privacy would be protected if the AOL and MSN query logs were collected, stored, and mined using CrowdLogging. To do this, we sort each query log by the anonymized user or session ID, so that we have all of a particular user's queries. We normalize all queries and remove duplicate queries that occurred in succession. We simulate the results of our algorithm on the assumption that each user is a client and there exists some server. We are interested in what information *could have* been discovered given the full log but is no longer available in our distributed and private approach.

Below, we show results for three applications: mining queries, query reformulations, and query-URL pairs. In each case, we analyze the effect that each of the four privacy policies described in Section 3.2 have on the log at large, as well as on one of the exposed AOL users.

There are several parameters that must be set for the differential privacy policies: δ, ϵ, d. Following Korolova et al. [16], we chose $\epsilon = \ln(10)$. Korolova et al. also suggest that $\delta < \frac{1}{n}$, where n is the number of users or sessions contained in the search log. We chose $\frac{1}{n+1}$ for either data set to maximize the amount of privacy that can be safely compromised. The differential privacy policies are constrained to only using d artifacts per user; we have decided to use the most recent d artifacts under the assumption that more recent artifacts are of higher interest than older ones. We used the largest d possible for each value of k.

4.1 Query frequency mining

Given the complete log, all queries can be counted, no matter how rare. We are interested in the portion of queries that are released under our framework as the anonymity threshold increases for the four privacy policies we discussed in Section 3.2. For the AOL log, we show this in Figure 3. The x-axis corresponds to the artifact or user threshold used for each policy. In the AOL logs, the FT_a and FT_u policies release a much larger fraction of the total query impressions compared with the other two policies. This fraction decreases rapidly and then slows, ending around 20% at $k = 160$. In contrast, the two differential privacy policies start off revealing a much smaller portion of total query impressions, but reveal more as k increases. This is because

[7]For example, http://www.atrus.org/hosted/AOL-data.tgz

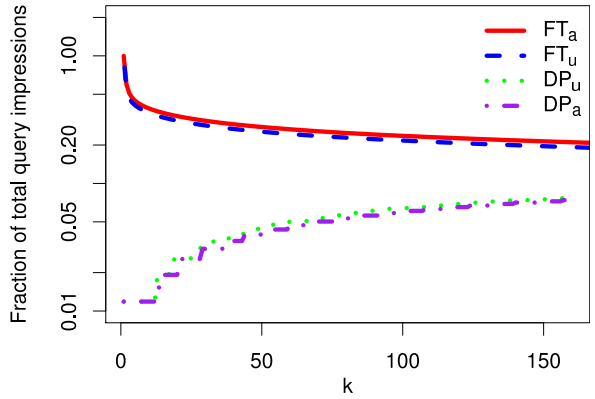

Figure 3: Proportion of total query impressions revealed in the AOL query logs as a function of the four privacy policies.

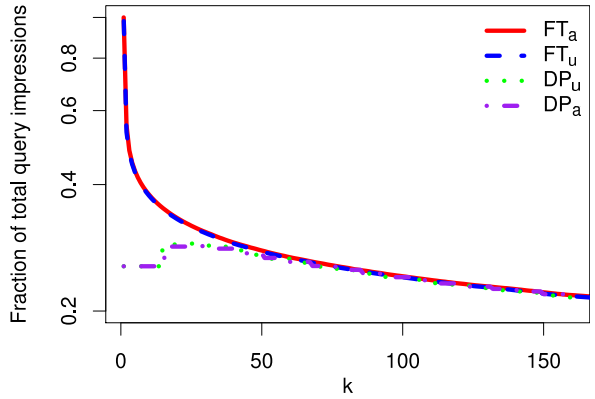

Figure 4: Proportion of total query impressions revealed in the MSN query logs as a function of the four privacy policies.

for any value of k, the differential privacy policies can only use a fixed d queries from each user. As k increases, d increases as well. The DP_u policy releases more queries than DP_a because, for a fixed k, the DP_u can use a larger d than DP_a can.

A similar graph for the MSN log is shown in Figure 4. FT_u and FT_a show a similar trend as on the AOL search log, although the differences between the two is less prominent than in the AOL logs. However, the two differential privacy policies show an initial spike and then follow a non-increasing trend, becoming in sync with the other two policies. This likely has to do with the session-based nature of the MSN log: sessions are generally short and include few queries. Once d exceeds the maximum number of queries in a session, the differential privacy policies are effectively the same as the other two policies. In addition, since sessions are short, it is less likely that a session will produce the same query multiple times, making the query and user frequencies virtually the same.

Due to space constraints, we do not show plots of the fraction of distinct queries under the different policies. For both logs, a very low portion of distinct queries is revealed and the trend under all policies is non-increasing as k increases.

To see how this loss affects an individual, we take a look at the queries of AOL user 4417749, the individual identified by the *New York Times* [4]. User 4417749 entered a total of 239 queries, 224 of which were distinct, meaning the user

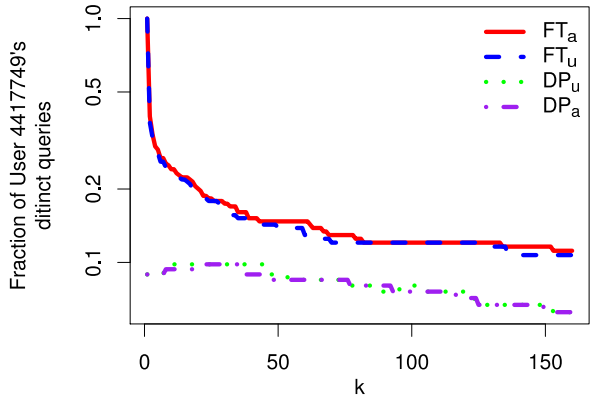

Figure 5: Proportion of distinct queries released for AOL user 4417749 under different privacy policies.

entered very few queries more than once. Figure 5 shows the proportion of the user's distinct queries revealed under each policy.

If we dive into user 4417749's queries and only concentrate on the non-differentially private policies, we see that highly identifiable queries, such as those referencing relatives' names (i.e., same surname) are discarded when $k > 1$. When $k > 2$, we begin to lose queries with location information, such as "gwinnett humane society" and "eugene oregon yellow pages". Queries released with $k \approx 20$ include possibly sensitive queries such as "mini strokes", "paranoia", and "loneliness", though recall that in our framework, those queries would not be associated with the user. When $k > 7500$, we begin to see only the most frequent and generic queries, such as "walmart", "yahoo", and "google".

The differential privacy policies are much more conservative. For a threshold of $1 \leq k \leq 7$, both policies are allowed only one query per user. At this level, both policies suppress virtually all sensitive queries, including queries that could give away location. As k increases, the only queries that remain are the extremely popular ones, like "google". The penalty is that we have access to at most 10% of the queries entered by user 4417749.

4.2 Query reformulation mining

Here we consider adjacent pairs of queries, though again we discard adjacent queries that are identical. We find about 18.7 million unique pairs with 20.4 million instances in the AOL log and 5.1 million distinct pairs with 5.5 million instances in the MSN log.

The portion of total query reformulation instances released from the AOL and MSN logs for varying levels of k under the four policies follow a very similar trend to releasing query impressions. For this reason, we do not provide the graphs. A major difference, however, is the scale: very few query reformulations are released for $k > 1$.

If we use a threshold of $k = 100$, we find that the privacy policies release 233K (FT_a), 196K (FT_u), 44K (DP_u), and 42K (DP_a) queries from the AOL logs—a very low proportion of the overall data set. For MSN, these numbers are smaller: 41k (FT_a), 32k (FT_u), 32k (DP_u), and 31k (DP_a). To have privacy preserved in reformulations, it appears we need to lower our privacy requirements or we need a larger log from which to draw query reformulations.

In the case of AOL user 4417749, we find 237 reformulations. This user never entered the same reformulation

twice, and the majority of the reformulations are unique to this user: at $k = 2$ there are only six reformulations that would be revealed under the FT_u policy, and only three reformulations occur more than a few times. Some of the reformulations are spelling mistakes ("lonelyness" → "loneliness", entered by one other user and "oggle" → "google", entered by 225 other users). One is a true conceptual reformulation: "jc penney" → "sears", entered by 46 other users. The "lonelyness" → "loneliness" reformulation is possibly the most sensitive of user 4417749's more widely occurring reformulations, but without knowing the source of reformulations, a researcher using the system would have no idea who entered it, nor would they be able to see what other broadly common queries or reformulations came specifically from the same user.

An interesting phenomenon occurs under the differential privacy policies: the most number of reformulations released by either is exactly one—"oggle" → "google"—and is only released for a short range of k, corresponding to $d = 3$. When $d = 3$, that particular reformulation occurred 22 times across 22 different users. For $d < 3$ and $d > 3$, the number of occurrences of this reformulation is always below the necessary k for both policies.

4.3 Query-URL frequency mining

Finally, we explore the effect of the four privacy policies on query-URL pair mining: how often is a search result URL clicked following a query? The AOL logs consist of 5.4 million distinct and 9.2 million total query-click pairs. The MSN logs have 1.6 million distinct and 2.4 million pairs. On average, each query has just under two associated clicks in both logs.

As k increases, the trend of the the four policies is similar to the one we see in the query frequency graphs. There are moderately fewer query-URL pairs released than query impressions in both search logs, but the difference is not as dramatic as with the query reformulation task. This is in part because there is only a limited set of possible clicks to follow a query, as opposed to the unbounded number of possible reformulations for a query. If k is low enough, the click information will be adequate for some re-ranking approaches under the FT_u and FT_a policies. Similar to the query frequency mining, there are more pairs released for the differential privacy policies as k increases.

AOL user 4417749's released query-URL pairs follow a trend similar to the query impressions released under each of the four policies for this user, though flatter. The user's log consists of 132 distinct and 136 total query-click pairs.

Many of the more popular query-click pairs entered by this user are for navigational queries, such as "google" → "http://www.google.com" and "jcpenny" → "http://www.-jcpenney.com", though some are clearly informational, like "sinus infection" → "http://www.emedicinehealth.com" and "termites" → "http://ianrpubs.unl.edu".

For differential privacy, only the navigational queries entered by AOL user 4417749 are released. At $k = 100$, DP_u and DP_a only release five navigational clicks for the queries: "google", "yahoo", "walmart", "costco", and "blue book" (as in Kelley Blue Book).

4.4 Conclusions of simulations

These simulations show that privacy policies with stronger guarantees have larger utility costs. Some tasks, such as query reformulation mining, will require larger logs for certain policies, such as DP_u and DP_a. However, we can also see that the privacy of the previously identified AOL user would likely have been protected under any of the policies simulated above, in large part due to the fact that artifact linking is explicitly prohibited in the released set.

5. PILOT USER STUDY

To evaluate the practicality of CrowdLogging, we conducted a pilot user study. In this section, we discuss many of the decisions we made in implementing the various components of the system. Then we describe the details of the study itself and provide an analysis of results from three applications we recently deployed on top of the framework. We end with a discussion of system vulnerabilities.

We stress that this study is small and only serves to validate the approach on real web browsers with real data. Collecting enough genuine log data for meaningful analysis is a substantial undertaking that is beyond the scope of this study—though we hope to do so in the future.

5.1 Implementation details

Our implementation, *CrowdLogger*, consists of four components: (1) an extension for the Mozilla Firefox Web browser[8] and (2) a client that together comprise the artifact mining and uploading mechanism; (3) one or more anonymizers, which implement a sender anonymization protocol based on onion routing [9]; and (4) a central server that aggregates the collected artifacts. All components other than the extension communicate using Java remote method invocation (RMI) [18]. We describe each of these components in more detail below.

Firefox extension. In addition to serving as CrowdLogger's user interface, the extension captures a log of the user's search and browsing activity. This process is similar to that of existing search engine toolbars, except that rather than sending log information to a central server, the browser stores it on the user's computer for later processing. The extension is also responsible for polling an application server for new mining tasks, and when available, invoking the client to execute the tasks. Users are prompted before mining tasks are run, giving them control over when their data is used.

Client. Upon being invoked by the extension, the client runs mining tasks on the user's log: it extracts search artifacts from the log, encrypts them using the AES symmetric cipher [6], generates a portion of the decryption key for each artifact by using Shamir's Secret Sharing [22], selects a path of anonymizers to the server, encrypts the encrypted artifacts using the server's and selected anonymizers' RSA public keys [20], and then sends the resulting items to the appropriate anonymizers. Encryption with the server's public key prevents anonymizers from viewing the encrypted search artifacts. The key used as the password for the symmetric cipher must be determined deterministically for each artifact; we set this to be a cryptographic hash value[9] of the artifact's plaintext normalized field, ensuring that every client will encrypt the same artifact in the same manner. The portion of the key passed along with each artifact is dependent on a user-selected pass phrase, thus ensuring that a user never

[8]http://www.mozilla.org/firefox/

[9]For example, using SHA1 [8]

inadvertently provides multiple key parts for the same artifact, even if they participate in experiments across multiple computers. While the network traffic caused by this process is considerable—linear in the number of artifacts—it is comparable to any query logging toolbar and far less than that caused by web services that give real time query suggestions or search results.

Anonymizers. The anonymizers function like the relays of the Tor anonymity network,[10] accepting messages from clients and other anonymizers, unpacking them by decrypting them with its private key, and sending the resulting messages to their next destination, which may be either another anonymizer or the server. As with other anonymizer-based methods, they work by directing traffic through several sites before finally sending it onto its destination. Although such approaches can be compromised with sufficiently advanced monitoring techniques [17] our main goal is that the server cannot extract location information, such as the originating IP address, from connections carrying arriving artifacts.

Server. The server collects anonymized, encrypted data from the clients as directed through the anonymizers. Whenever the server receives a sufficient number of distinct key portions for a given search artifact, it reconstructs the complete key using Shamir's Secret Sharing and decrypts the artifact. The aggregation occurs as described in Section 3.1 for all of the applications we have implemented so far.

5.2 Study details

Once our implementation was complete, we advertised our pilot study to associates via e-mail. Users were asked to accept an informed consent form and were then directed to download the Firefox extension. Upon installing, we asked them to create a user ID or use their existing one, if they had one, and then to register. (The user ID is not associated with any identifying information, so privacy is preserved.) Note that a user could install the extension on multiple computers and register at each location, though the registrations were not linked. The main purpose of the registration was to collect demographic information about the users in the study; it is not part of the framework. The compensation for the study consisted of a weekly drawing for a $20 Amazon.com gift card. Participants were given a greater chance of winning if they referred friends to the study. (Prizes were awarded anonymously since we did not collect participant contact information.)

At the time of submission, the Firefox extension had been downloaded and installed 40 times, with 31 registrations received (some of which were due to the authors).

Our institutional review board approved a version of the system that uses k-anonymity (privacy policy FT_u) with k=5. That means a search artifact must be issued under at least five *distinct* user IDs for it to be decrypted at the server. We are using this study to evaluate our implementation. The pilot study ran for two weeks in January 2011.

5.3 Experiments

We ran the three applications we used for our simulations: query frequency, query reformulation, and query-URL pair mining. For each of the experiments, we compare the four privacy policies to understand what data would be released under each model. Note that most of the data is actually encrypted. However, in our study setup, we know how many people entered a particular artifact and how many instances occurred. The fact that we do not know what the artifact is is inconsequential for our immediate purposes.

[10]http://www.torproject.org/

	Query freq.		Query reform.		Query-click	
k	FT_a	FT_u	FT_a	FT_u	FT_a	FT_u
1	4905	4905	6194	6194	2816	2816
2	1803	46	1783	12	830	9
3	490	13	118	1	288	1
4	402	6	89	1	209	0
5	156	1	36	0	126	0

Table 1: Number of distinct search artifacts released under the FT_u and FT_a privacy policies for three applications run in the implemented framework.

A total of 16 distinct users participated in these experiments from 23 installations. No artifact is shared by more than 8 users, but the trends are evident at $k = 5$, so we analyze k from 1–5. For the differential privacy policies, we use $n = 16$ and $\epsilon = 1/(n + 1)$.

For the query frequency task, 8363 impressions were mined of which 4905 are distinct. A total of 8305 query reformulation instances were extracted from the logs; 6194 were distinct. The query-click pair mining task resulted in 5432 pairs, 2816 of which were distinct. A summary of the amount of data released under the FT_a and FT_u policies for the three applications is shown in Table 1. There is no threshold for which information would be released under either differential privacy method—there were simply too few users for this experiment. For this reason, they are omitted from the table.

The results show that it is possible to implement this system with low overhead on the participants' browsers. Information was collected, aggregated, and decrypted when possible (though our set was small enough that only a control "dummy" query (not included in the statistics in Table 1) and the query "news" appeared often enough to become visible). We are planning a larger study in the future, with support for browsers in addition to Firefox.

5.4 System vulnerabilities

As with any query logging approach, CrowdLogging is vulnerable to certain types of malicious attacks. Attackers could potentially compromise users' computers and steal their search logs, take over one or more of the anonymizers, take over the server, or impersonate user nodes.

The first case applies to all software. If an attacker has access to a user's computer, there is likely much more at stake than the user's search data.

If one or more anonymizers are compromised, the attacker cannot do much, as all of the data is encrypted using the server's RSA public key. If both the anonymizers and the server were attacked simultaneously, then the anonymizers could be re-programmed to keep IP addresses associated with incoming artifacts, potentially revealing users.

If the server is compromised, then a brute-force probing attack can be launched against the artifacts encrypted using the secret sharing scheme. To do this, the attacker must generate a candidate artifact, encrypt it using the sharing scheme, and then compare it with the list of encrypted artifacts. If a match is found, the attacher knows that the candidate artifact is present in the collected data. However,

this process is not trivial, so it is unlikely that even a large portion of the collected data would be compromised.

Finally, if attackers impersonate some number of user nodes, they can inject identical artificial data into their logs which may cause specific artifacts to be released that would not otherwise have been released. For example, perhaps the query "bob" was only entered by one legitimate user. If $k = 5$ and there are four attackers that are curious if any other users entered the query "bob", they could add that search to their logs. The artifact "bob" now appears to come from k distinct users and is thus revealed. There is no easy solution to this attack, though we are exploring techniques that might indicate when such an attack is occurring, such as by allowing for trusted users to register and then checking for skew between data submitted by registered and unregistered users.

6. CONCLUSIONS

We presented a novel framework for collecting, storing, and mining search logs in a distributed, private, and anonymous manner called CrowdLogging. The work is motivated jointly by a need to have search logs available to researchers outside of large search companies and a need to instill trust in the users that provide search data. Our framework gives researchers access to up-to-date search data. CrowdLogging also allows users—the individuals who contribute the data that is ultimately mined—an unprecedented amount of control over their data.

We used simulations on two large search logs to show the practicality of several of the privacy policies that are compatible with our framework, including two with theoretically sound privacy guarantees. The simulations show that much of the original data can be ultimately revealed, although the exact amount varies by application and privacy policy.

We showed the viability of the system through a small pilot user study, run over two weeks. This study required a fully functional implementation, which we named CrowdLogger, and demonstrated that all parts of the system work with minimal overhead. A larger study is planned for the future and will further assess the feasibility of using CrowdLogging in realistic settings.

7. ACKNOWLEDGMENTS

We thank Brian Levine for his helpful thoughts about system vulnerabilities and possible defenses. This work was supported in part by the Center for Intelligent Information Retrieval, in part by NSF grant #IIS-0910884, and in part by an NSF Graduate Research Fellowship under grant #DGE-0907995. Any opinions, findings and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect those of the sponsor.

References

[1] E. Adar. User 4xxxxx9: Anonymizing query logs. In *Query Logs Workshop, WWW*, 2007.

[2] F. Ahmad and G. Kondrak. Learning a spelling error model from search query logs. In *HLT '05*. ACL, 2005.

[3] J. Bar-Ilan. Position paper: Access to query logs—an academic researcher's point of view. In *Query Log Analysis Workshop, WWW*, 2007.

[4] M. Barbaro and T. Zeller Jr. A face is exposed for AOL searcher No. 4417749. *New York Times*, 2006. `http://www.nytimes.com/2006/08/09/technology/09aol.html`.

[5] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *KDD '00*. ACM, 2000.

[6] J. Daemen and V. Rijmen. *The design of Rijndael: AES–the advanced encryption standard*. Springer Verlag, 2002.

[7] C. Dwork. Differential privacy. *Automata, languages and programming*, pages 1–12, 2006.

[8] P. FIPS. 180-3 Secure Hash Standard (SHS). *Information Technology Laboratory, NIST, Oct*, 2008.

[9] D. Goldschlag, M. Reed, and P. Syverson. Onion routing. *CACM*, 42(2):39–41, 1999.

[10] K. Hafner and M. Richtel. Google resists U.S. subpoena of search data. *New York Times*, 2006. `http://www.nytimes.com/2006/01/20/technology/20google.html`.

[11] Y. Hong, X. He, J. Vaidya, N. Adam, and V. Atluri. Effective anonymization of query logs. In *CIKM '09*, 2009.

[12] D. Howe and H. Nissenbaum. *TrackMeNot: Resisting Surveillance in Web Search*. Oxford Univ. Press, 2009.

[13] B. J. Jansen and A. Spink. How are we searching the World Wide Web? A comparison of nine search engine transaction logs. *Inf. Process. Manage.*, 42(1):248–263, 2006.

[14] B. J. Jansen, A. Spink, and J. Pedersen. A temporal comparison of AltaVista web searching. *JASIST*, 56: 559–570, 2005.

[15] T. Joachims. Optimizing search engines using clickthrough data. In *Proc. of the eighth ACM SIGKDD*, pages 133–142. ACM, 2002. ISBN 158113567X.

[16] A. Korolova, K. Kenthapadi, N. Mishra, and A. Ntoulas. Releasing search queries and clicks privately. In *Proc. of WWW*, pages 171–180, 2009.

[17] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *IEEE Security and Privacy Symposium*, 2005.

[18] Oracle. Java RMI spec. `http://download.oracle.com/javase/6/docs/platform/rmi/spec/rmiTOC.html`.

[19] M. Paşca. Weakly-supervised discovery of named entities using web search queries. In *CIKM '07*. ACM, 2007.

[20] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[21] M. J. Schwartz. Spam attack captures government data. *Information Week*, 2011. `http://www.informationweek.com/news/security/attacks/showArticle.jhtml?articleID=229000118&cid=RSSfeed_IWK_All#`.

[22] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[23] L. Sweeney. k-Anonymity: A Model for Protecting Privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst*, 10(5):557–570, 2002.

[24] S. Wedig and O. Madani. A large-scale analysis of query logs for assessing personalization opportunities. In *KDD '06*. ACM, 2006.

[25] J.-M. Yang, R. Cai, F. Jing, S. Wang, L. Zhang, and W.-Y. Ma. Search-based query suggestion. In *CIKM '08*, 2008.

[26] S. Zhong, Z. Yang, and T. Chen. k-anonymous data collection. *Information Sciences*, 179(17):2948–2963, 2009.