On Anonymizing Query Logs via Token-based Hashing

Ravi Kumar Jasmine Novak Bo Pang Andrew Tomkins
Yahoo! Research
701 First Ave
Sunnyvale, CA 94089.
{ravikumar,jnovak,bopang,atomkins}@yahoo-inc.com

ABSTRACT

In this paper we study the privacy preservation properties of a specific technique for query log anonymization: token-based hashing. In this approach, each query is tokenized, and then a secure hash function is applied to each token. We show that statistical techniques may be applied to partially compromise the anonymization. We then analyze the specific risks that arise from these partial compromises, focused on revelation of identity from unambiguous names, addresses, and so forth, and the revelation of facts associated with an identity that are deemed to be highly sensitive. Our goal in this work is twofold: to show that token-based hashing is unsuitable for anonymization, and to present a concrete analysis of specific techniques that may be effective in breaching privacy, against which other anonymization schemes should be measured.

Categories and Subject Descriptors

H.3.m [**Information Storage and Retrieval**]: Miscellaneous

General Terms

Algorithms, Experimentation, Measurements

Keywords

Query logs, privacy, hash-based anonymization

1. INTRODUCTION

On July 29, 2006, AOL released over twenty million search queries from over 600K users, representing about 1.5% of AOL's search data from March, April, and May of 2006. The data contained the query, session id, anonymized user id, and the rank and domain of the clicked result. The media field day began almost immediately, with journalists competing to identify the most scandalous and revealing sessions in the data. Nine days after the release, AOL issued an apology and called the release a "screw up," removed the web site, and terminated a number of employees responsible for the decision, including the CTO.

There is great appetite to study query logs as a rich window into human intent, but as this vignette shows, the privacy concerns are broad and well-founded, and the public is rightly sensitive to potential breaches. Academic researchers are enthusiastic about receiving anonymized data for research purposes, but to date, there is no satisfying framework for proving privacy properties of a query log anonymization scheme. We do not have such a framework to propose. Instead, we present a practical analysis of a natural anonymization scheme, and show that it may be broken to reveal information broadly considered to be highly sensitive.

The particular scheme we study is *token-based hashing*, in which each search string is tokenized, and each token is securely hashed into an identifier. We show that serious leaks are possible in token-based hashing even when the order of the underlying tokens is hidden. Our basic technique is the following. We assume the attacker has access to a "reference" query log that has been released in its entirety, such as the AOL query log, or earlier logs released by Excite or Altavista. We employ the reference query log to extract statistical properties of words in the log-file. We then process the anonymized log to invert the hash function based on co-occurrences of tokens within searches; interestingly, inverting cannot be done using just the token frequencies.

The technical matching algorithms we employ must provide good accuracy while being somewhat efficient to run on large query logs. This turns out to be a nontrivial problem, and much of our time is spent describing and evaluating our approaches to address this efficiency issue.

1.1 The sensitivity of revealed data

Based on the mapping extracted between hashes in the anonymized query log and words in the reference query log, we perform a detailed evaluation of the potential for uncovering sensitive information from a log protected by token-based hashing. Where possible, we incorporate publicly-available third-party information that would be available for an attacker. We begin by focusing on person names, which are particularly sensitive due to the large number of "vanity queries" that occur in log-files, in which a user searches for his or her own name. We study extraction of these names using a hand-built name spotter seeded with a list of common first and last names, employing public data from the US census to help in the matching.

Surprisingly, we are aided in matching obscure names by the prevalence of queries for celebrities. By matching the co-occurrence properties of "Tom Cruise" or "Jane Fonda," we learn the hash values corresponding to the first names Tom and Jane. From there, we will miss last names that are unique and unambiguous, but we will capture many other last names that occur in other contexts with characteristic co-occurrence properties.

Copyright is held by the International World Wide Web Conference Committee (IW3C2). Distribution of these papers is limited to classroom use, and personal use by others.
WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

We also study the extraction of locations (particularly city/state pairs), company names, adult searches, and *revealing terms* that would be highly sensitive if published. Revealing terms include pornographic queries, as well as queries around such topics as murder, suicide, searches for employment, and the like.

We study the number of sessions containing a properly extracted relatively non-famous name and one of these other categories of terms. We unearthed numerous sessions containing de-anonymized names of non-famous individuals along with queries for adult and revealing terms.

In the context of query logs released without session information, there are two primary risks. First, there are many techniques to try to re-establish session links by analyzing query term and topic similarity. Second, we also unearthed various de-anonymized queries containing both a reference to a non-famous person and a location.

1.2 Comments on our approach

There have been numerous approaches to defining a framework capturing what is meant by "privacy." In this work, we argue that attackers will naturally make use of significant amounts of domain knowledge, large external corpora, and highly tailored approaches. In addition to work on frameworks and provable guarantees, usefully anonymized query logs will need to be scrutinized from the perspective of a sophisticated attacker before they can be released; at the very least, they should be proof against the attacks we describe.

That said, we should also note that our approach contains a key weakness. Specifically, it allows us to find only terms that exist in the reference query log, and that occur in the anonymized query log with sufficiently rich co-occurrences. Sequences of digits, such as street numbers, are very unlikely to be matched unless they occur in another context (such as a very famous address, the name of a song, common model number, or the like).

2. RELATED WORK

There is a large and thriving body of work on search log analysis, which has resulted in highly valuable insights in many different areas, including broad query characterization [17, 3, 14, 4], broad behavioral analysis of searches [4, 5, 19], deeper analysis of particular query formats [20, 21], query clustering [15], term caching [9], and query reformulation [6]. In every one of these cases, anonymization at the level of the query as provided by a hash of the entire search string would have made the analysis impossible. In all cases but the query format analysis, token-based hashing would have allowed some interesting work, and in most cases, the entire research agenda would have been admissible. Thus, there are many arguments in favor of token-based hashing as an approach to anonymization. We present the flip side of the coin, with an analysis of the dangers, and we conclude that significant privacy breaches would occur.

The best-studied framework for privacy preservation is k-anonymity, introduced by Samarati and Sweeney [16], and studied in a wide range of follow-on work (see for instance [2, 10, 22] and related work). The model is stated in terms of structured records. A relation is mapped row-by-row to a new privacy-preserving relation, which is said to be k-anonymous if each set of potentially revealing values (for instance, the zip code, age, and gender of an individual) occurs at least k times. The motivation behind the definition is as follows: even if there are external sources that might allow mapping of such indirect data as zip code, age, and gender back to a particular individual, nonetheless, the new anonymized database will map back to at least k different individuals, providing some measure of privacy for sufficiently large k. There are two concerns with this scheme in our world. First, our setting is not naturally structured, so it is unclear how to extend k-anonymity; it is clearly not practical to make the entire session history of a user identical to that of $k-1$ other users. In fact, it is not clear which parts of a query session should even be treated as values in a relation. And second, revealing that somebody in a set of one hundred users is querying about techniques for suicide is already revealing too much information.

The related problems of text-based pseudonym discovery and stylometrics have been heavily studied; in these problems a body of text is available from a number of authors, and the goal is to determine which of these authors are identical. See [11] and the references therein. The problem of aligning hashes in one log file with tokens in another also resembles previous work in statistical machine translation that automatically construct bilingual lexicon (dictionary) from parallel corpora (text in one language together with its translation in the other language). If we look more closely, they are very different beyond the resemblance at the surface level. Most notably, while work in bilingual lexicon construction in machine translation assumes sentence-level alignment in the parallel corpora, we do not have query-level alignment between the two log files; furthermore, the two log files are very far from being semantically equivalent.

There is a large body of work on log anonymization; see for instance [12, 18]. This problem is superficially related to ours, but the techniques used are very different. The goal is to provide anonymity, but classical approaches focus on hiding the IP address, while later approaches propose developing application-dependent multiple levels of privacy for a much wider set of attributes. Nonetheless, the problems that arise in our domain of mapping large numbers of words based on an enormous co-occurrence matrix do not arise in anonymization of network logs.

Our formulation of the problem is also somewhat related to the well-known problem of graph matching and graph isomorphism. The difference, however, is that our graphs are richer in terms of what they represent and so are more amenable to statistical techniques.

3. MAPPING ALGORITHMS

We begin with some notation, and a formal definition of our problem. We then give an overview of the dataset we will study. With data in hand, we describe our family of algorithms and give performance results comparing them. In the following section, we will turn to a discussion of the results themselves, and cover the privacy implications in more detail.

3.1 Preliminaries

We begin with some notation. Recall that we will employ an unhashed query log in order to generate statistics for our attack on the hashed query log. Let Q_R be the raw (unhashed) query log and Q_A be the anonymized (hashed) query log.

For a query log Q (raw or anonymized), let term(Q) denote the set of all terms that occur in Q; in the case when Q

is a raw query log, this will be the set of *tokens* and when Q is an anonymized query log, this will be the set of *hashes*. Let $\text{freq}(s, Q)$ denote the number of times the term s occurs in Q; let $\text{freq}_N(s, Q) = \text{freq}(s, Q)/\sum_t \text{freq}(t, Q)$ be its normalized version corresponding to the probability of s in log Q. Let $\text{cooc}(s, t, Q)$ denote the number of times s co-occurs with t in Q; let $\text{cooc}_N(s, t, Q) = \text{cooc}(s, t, Q)/\sum_{t'} \text{cooc}(s, t', Q)$ be its normalized version, representing the probability that a particular term co-occurring with s is in fact t. We drop Q whenever the query log is clear from the context.

Recall that our goal is to map the hashes of Q_A to the tokens of Q_R. We will employ a bipartite graph to reflect the candidate mappings between hashes and tokens, as follows. Define a weighted bipartite graph $G = (L, R, E)$ as a set of left nodes L, right nodes R, and edges $e = (\ell, r) \in E \subseteq L \times R$. By convention, we will always take L to be a set of hashes, and R to be a set of tokens. Let $w : E \to \mathbb{R}$ be a real-valued weight function on edges; $w(e)$ will represent the quality of the map between the hash and the token connected by edge e.

Fix a vocabulary size n, and let $G_n = (L_n, R_n, E_n)$ be a bipartite graph representing mappings between the most frequent n hashes in Q_A and the most frequent n tokens in Q_R. Our goal is to map the hashes in L_n to tokens in R_n, taking into account $\text{freq}(\cdot)$ and $\text{cooc}(\cdot)$ information; in other words, we seek a bijective mapping $\mu : L_n \to R_n$.

Accuracy and matchable sets.

We define the following performance metric of a mapping μ for a vocabulary size n. Given L and R, let $\mu^* : L \to R \cup \{\perp\}$ be the correct mapping of hashes to tokens, where the function takes $\perp$ if the hash has no corresponding token on the right hand side. Given a mapping $\mu : L \to R$, the *accuracy* is defined to be

$$\frac{|\{\ell \mid \mu(\ell) = \mu^*(\ell)\}}{|\{\ell \mid \mu^*(\ell) \neq \perp\}|}.$$

The denominator of this expression is the size of the *matchable set*, which is the set of hashes that can possibly be mapped to tokens. This set imposes an upper bound on the performance of any mapping and therefore, accuracy measures the fraction of the matchable set obtained by μ. In our results, we specify the accuracy and wherever applicable, the size of matchable set.

High-level approach.

We use the following general framework to compute the mapping. Our framework can be expressed in terms of how two generic functions, namely, InitialMapping and UpdateMapping, are realized.

Algorithm ComputeMapping (Q_A, Q_R, n)

$\mu \leftarrow$ InitialMapping (Q_A, Q_R)

While not done

$\mu \leftarrow$ UpdateMapping (Q_A, Q_R, μ)

The function InitialMapping takes L, R along with the query logs and computes an initial candidate mapping $\mu : L \to R$. The function UpdateMapping takes L, R, the query logs, and the current mapping, and outputs a new mapping. Based on different realizations of these functions, we obtain different methods for computing the mapping.

Data.

We use log files from Yahoo! web search in our experiments. For privacy reasons, these files are carefully controlled and cannot be released for general study (especially under token-based hashing). In general, we extract one set of queries to act as the raw log Q_R, and a distinct set of queries to act as the anonymized log Q_A. We process the anonymized log file by performing white-space tokenization, and applying a secure hash function to each token, producing hashes that we must now try to invert. For all the experiments in this section, the query log files consist of random samples of six-hour query logs from a week apart in May, 2006. Each log contains about 3 million queries in total. In Section 4 we will consider other log-file pairs.

3.2 Choosing an initial mapping

We study three approaches to selecting an initial mapping, as follows:

Random. Randomly assign each node in L to a unique node in R in a one-to-one fashion.

Frequency-based. Order the hashes in L by $\text{freq}(\cdot, Q_A)$ and the tokens in R by $\text{freq}(\cdot, Q_R)$. Then the i-th most frequent hash in L is mapped to the i-th most frequent token in R.

NodeStart. This is a more complex technique that builds a simple five-element vector combining different types of information about a token or a hash. All five elements of this vector can be computed on a completely hashed query log, and thus represent a fingerprint of the style in which the token or hash appears in the log. If a hash and a token have very different fingerprints, then the hash is unlikely to have been computed from that token. The five dimensions of the feature vector $g(s)$ are:

1. The normalized frequency, $\text{freq}_N(s, Q)$.
2. The number of times s appeared as a singleton query in Q, divided by $\text{freq}(s, Q)$.
3. The co-occurrence count, $\sum_t \text{cooc}(s, t, Q)$.
4. The neighbor count, $|\{t \mid \text{cooc}(s, t, Q) > 0\}|$.
5. The average normalized frequency given by, $(\sum_t \text{freq}_N(t, Q) \cdot \text{cooc}(s, t, Q))/(\sum_t \text{cooc}(s, t, Q))$.

We compute this feature vector for each node in L and then normalize the values of each dimension to have mean 0 and standard deviation 1. Similarly, we compute a normalized feature vector for each node in R, where the normalizations are dependent on the other values of R. The distance between two nodes $\ell \in L$ and $r \in R$ is simply the L_1 distance between their vectors: $|g(\ell) - g(r)|$.

The initial mapping is then computed by the score-based greedy, described in Section 3.3.1; for now, it suffices to assume that this method computes a mapping of hashes to tokens using the L_1 distance we computed. We evaluate all of these initial mappings in the context of a greedy UpdateMapping function described below in Section 3.3. Figure 1 shows the results for each of the three InitialMapping functions just described, for various different iterations of the UpdateMapping function. The figure clearly shows

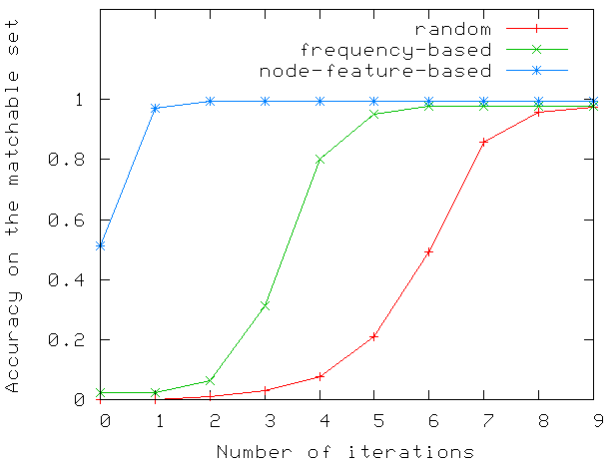

Figure 1: Accuracy for vocabulary size $n = 1000$ ($|$matchable set$| = 918$) using different initial mappings.

that the accuracy is almost independent of the choice of initial mappings. However, the sophisticated NodeStart mapping reaches the maximum accuracy quite quickly. Moreover, the accuracy of the frequency-based mapping at iteration 0 hints that it is hopeless to just use the frequencies of the hashes and the tokens towards obtaining a mapping. It is also interesting to observe the 'S'-shaped curve corresponding to the random initial mapping; this suggests the presence of a threshold at which point the randomness in the initial mapping is slowly replaced by structure. Note that the plateau at the end of the NodeStart curve does not reflect a stable mapping. Although the accuracy stays the same starting from iteration two, portions of the mapping are still changing at each iteration. All further experiments employ the NodeStart function.

3.3 Updating the mapping

This section describes various different approaches to updating the mapping. However, to begin, we discuss the general problem of comparing various candidate tokens as mappings for a particular hash, based on information in the current mapping.

Figure 2 gives an example of the situation that may arise when computing the distance between a hash and a word. We wish to evaluate the quality of the map between hash h and token w. The mapping has already mapped h_1 to w_2, and h_3 to w_3, so the distance computation should take this mapping into account: h and w share co-occurrences. If later information becomes available about the other unmapped neighbors of h, the distance between h and w will need to be updated.

Distance measure. The distance measure we adopt is the following. We represent each node as a vector over $|L|+|R|$ dimensions whose entries give the co-occurrence probabilities with the corresponding token or hash. Tokens have non-zero weight only in dimensions corresponding to tokens. Hashes begin with non-zero weight only in dimensions corresponding to hashes, but each time a hash h is mapped

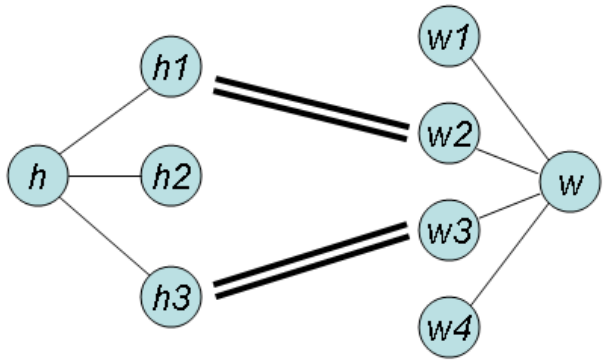

Figure 2: A candidate mapping between a hash and a token.

to a token w, all non-zero entries for h are migrated to w. In Figure 2, for instance, hash h will have non-zero entries only for h_2, w_2, and w_3. Distance is then given by the L_1 distance between the corresponding vectors.

Mapping-based distance. Rather than actually perform this migration of non-zero entries, however, we simply define the distance in terms of the initial co-occurrences among hashes and among tokens, based on a mapping function μ, as follows:

$$d_\mu(\ell, r) = \sum_{\ell' \in L} |\text{cooc}_N(\ell, \ell', Q_A) - \text{cooc}_N(r, \mu(\ell'), Q_R)|.$$

This idea falls within the general theme of identifying similar tokens through similar contexts. For instance, based on this intuition, past work has explored word clustering [13] and paraphrase extraction [1] using natural language texts from a single language. We differ from such previous work in that a mapping between hashes and tokens is involved in defining the distributional similarity. In addition, the kind of contexts at our disposal (co-occurring words within queries) can be very different from the kind of contexts available from proper, grammatical English sentences.[1]

We compare L_1 measure against corresponding quantities for L_2 and cosine measures, using the following method. We pick $n = 10,000$ and we choose a random sample of 1000 hashes. For each hash, we order the tokens according to either L_1, L_2, or cosine measures. The fraction of times the closest token under the measure was indeed the correct token is shown in the table below.

L_1	L_2	Cosine
0.93	0.75	0.8

This shows that L_1 measure clearly dominates the other measures. Note that this is in line with the result of Lee [8] who showed that L_1 measure is preferred over L_2 and cosine

[1]Note that although this prevents us from getting fine-grained contexts via syntactic analysis of full-length sentences, we may be getting an approximation of the optimal context by using all other words appearing in the same query as the context for the target word. After all, users are more likely to type in the "essential" words, which can be viewed as a "distilled" version of what the corresponding sentence would have been.

measures for such scenarios. Hence, we adopt L_1 measure as our distance going forward.

We now turn to schemes for UPDATEMAPPING. We present four schemes, the first two based on a distance score, and the last two based on post-processing of the distance score to produce a ranked list of candidates for each hash and each token.

3.3.1 Score-based methods

We discuss two score-based methods — greedy and a method based on the minimum cost perfect matching.

SCORE-BASED GREEDY. In the score-based greedy method, we consider all pairs $\ell \in L, r \in R$ and sort them by the distance $d_\mu(\ell, r)$. We then maintain the $L \times R$ triples $\langle \ell, r, d_\mu(\ell, r)\rangle$ on a heap. At each step, we pick the triple $\langle \ell, r, d\rangle$ with the minimum d value from the heap, set the updated mapping $\mu'(\ell) = r$, and delete all elements in the heap of the form $\langle \ell, \cdot, \cdot\rangle$ and $\langle \cdot, r, \ell\rangle$. The running time of this greedy method is $O(n^2 \log n)$, where the running time is dominated by having to compute all the pairwise distances. For the rest of the paper, greedy will always refer to the score-based greedy.

MINIMUM COST PERFECT MATCHING. Instead of constructing the mapping in a greedy way using scores, we can appeal to the minimum cost perfect matching formulation, applied to the bipartite graph with $w(\ell, r) = d_\mu(\ell, r)$. Recall that in minimum cost perfect matching, the goal is to find a bijective map $\mu' : L \to R$ that minimizes $\sum_{\ell \in L, r \in R} d_\mu(\ell, r)$. Using standard algorithms, this problem can be solved in time $O(n^{5/2})$ (see [7]). The solution to this problem yields the updated mapping μ'.

3.3.2 Rank-based methods

We discuss two rank-based method — greedy and a method based on the stable marriage problem.

RANK-BASED GREEDY. In the rank-based greedy method, we use the rank information instead of the score information. Formally, the function $d_\mu(\ell, \cdot)$ provides a ranking of all $r \in R$ with respect to ℓ; let the rank of $r \in R$ be $\text{rank}_\ell(r)$. Likewise, the function $d_\mu(\cdot, r)$ can be used to obtain the rank of $\ell \in L$ with respect to r, denoted $\text{rank}_r(\ell)$. Let $d(\ell, r) = \text{rank}_\ell(r) + \text{rank}_r(\ell)$. We now apply the score-based greedy method with the above distance function $d(\cdot, \cdot)$ to find the updated mapping μ.

STABLE MARRIAGE. Recall the stable marriage problem. We are given a bipartite graph consisting of men and women, where each man ranks all the women and each woman ranks all the men. A marriage (bijective matching) of men and women is said to be unstable if there are two couples (m, w) and (m', w') such that m ranks w' above w and w' ranks m above m'. Given the bipartite graph, the goal is to construct a marriage that is stable. This problem can be solved in $O(n^2)$ time (see [7]).

In our case, the men correspond to L and the women correspond to R and as in the rank-based greedy case, the function $d_\mu(\ell, \cdot)$ provides a ranking of all $r \in R$ with respect to $\ell \in L$ and the function $d_\mu(\cdot, r)$ provides a ranking of all $\ell \in L$ with respect to $r \in R$. Hence by applying the stable marriage algorithm, we can find the updated mapping μ.

Table 1 shows the results. The performance of score-based greedy is on par with the other three algorithms and since score-based greedy is simpler, we use this method going forward.

3.4 Efficiency considerations

The methods presented in the previous section take quadratic time to run. For increasingly deep query logs, it is not possible to proceed without some modifications for efficiency. We describe a number of approaches here.

3.4.1 Expanding the vocabulary using distance approximations

In this approach we use a fixed $\mu : L \to R$ to approximate the distance between a hash $\ell' \in L' \supset L$ and a token $r' \in R' \supset R$. Let $g'(\cdot)$ be the NODESTART function for the larger vocabulary. Let

$$\gamma(\ell') = \sum_{\ell \in L} \text{cooc}'_N(\ell', \ell, Q_A)$$

be the mass of the co-occurrences covered by μ. Let

$$\rho(\ell', r') = \sum_{\ell \in L} \min(\text{cooc}'_N(\ell', \ell, Q_A), \text{cooc}'_N(r', \mu(\ell), Q_R))$$

be the overlap between ℓ' and r' within L. Let

$$\delta(\ell', r') = 1 - \frac{\rho(\ell', r')}{\gamma(\ell')}$$

be an estimate of the distance between ℓ' and r' as given by the terms in the size n vocabulary. We then set the distance

$$\tilde{d}'(\ell', r') = \delta(\ell', r') + (1 - \gamma(\ell')) \cdot \frac{|g'(\ell') - g'(r')}{C},$$

where C is set to the number of features (in our case, 5). Note that if $\gamma(\ell')$ is bounded away from 0, then $\delta(\ell', r')$ is perhaps a good estimate of the actual distance and the first term dominates and if $\gamma(\ell')$ is close to 0, then $g'(\cdot)$ plays a heavier role.

We will report some experiments for this expansion after describing a pruning technique below.

3.4.2 Pruning

We give two approaches to heuristic pruning that can dramatically reduce the number of candidate hash-token pairs that must be considered.

α-PRUNING. The first approach is to restrict the set of pairs $(\ell, r) \in L \times R$ that are ever considered in all the UPDATEMAPPING methods. For each ℓ, we order the r's based on increasing values of $|g(\ell) - g(r)|$ and choose the top α fraction of r's in this ordering, for some $\alpha < 1$. Thus, the total number of pairs to be considered is now αn^2.

β-PRUNING. In a similar spirit, for each s, we choose $T' \subseteq \text{term}(Q)$ such that $|T'|$ is minimal and $\sum_{t' \in T'} \text{cooc}_N(s, t, Q) \geq \beta$, for some $\beta < 1$. In other words, each s chooses the fewest t's such that these t's garner at least β mass of the co-occurrence distribution. We do not explore the performance of β-pruning further.

To postulate the effect of pruning, we study how far the ranks of hashes and tokens migrate. Let $\text{rank}(s, Q)$ be the rank of s, when the terms in Q are ordered according to $\text{freq}(s, Q)$. Specifically, for $\ell \in L$, we plot the distribution of

Vocabulary (n)	matchable set	score greedy	mincost matching	rank greedy	stable marriage
1000	918	0.99	0.99	0.99	0.99
2000	1851	0.96	0.96	0.97	0.96
4000	3648	0.92	0.92	0.91	0.92
8000	7182	0.83	0.85	0.82	0.83

Table 1: Accuracy of score-based greedy, mincost matching, rank-based greedy, and stable-marriage algorithms.

$|\text{rank}(\ell, L) - \text{rank}(\mu^*(\ell), R)|$, where μ^* is the correct mapping. Figure 3 plots this value for various buckets of values of $\text{rank}(\ell, L)$. For example, an x-value of 200 corresponds to tokens with rank from 100 to 200. And the y-axis shows the absolute distance between the token's rank in Q_R versus Q_A.

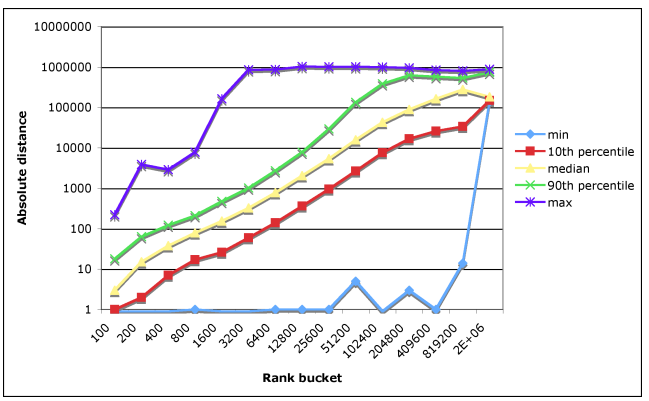

Figure 3: Rank migration.

We present a simple evaluation of the effectiveness of α-pruning and vocabulary expansion. We study a 2K-word mapping problem using the most frequent terms of our query logs. The size of the matchable set for this case is 1851, so we measure performance as number of correctly mapped hashes out of 1851. We perform four experiments.

Exp1: Begin by mapping 1K nodes using NodeStart and 10 iterations of greedy updates. Then perform vocabulary expansion with α-pruning using $\alpha = 0.1$ in order to map the remaining 1K nodes.

Exp2: Begin by mapping 1K nodes using NodeStart and 10 iterations of greedy updates. Then perform vocabulary expansion with no α-pruning in order to map the remaining 1K nodes.

Exp3: Begin by mapping 2K nodes using NodeStart, then perform a single iteration of greedy updates.

Exp4: Begin by mapping 2K nodes using NodeStart, then perform two iterations of greedy updates.

The success rates are as follows.

Experiment	1	2	3	4
Accuracy	0.96	0.98	0.94	0.98

Thus, α-pruning shows some impact on overall performance, but this cost may be acceptable at a 10X improvement in runtime. Vocabulary expansion is capable of high accuracy, and is thus a promising technique for larger problems scales. We employ this technique for larger runs in Section 4.

We now present two additional approaches to improving efficiency, each of which may be employed in either an exact setting or an approximate setting for greater efficiency. The first is based on a heap structure for continuous update of the possible mappings, and the second is based on an inverted index. We present these approaches, and have implemented them in our algorithms, but we leave a thorough performance evaluation for future work.

3.4.3 *Heap-based continuous update*

In the first proposal, we continuously enlarge the domain of μ and use this to approximate the distance between a hash $\ell' \in L' \setminus L$ and a token $r' \in R' \setminus R$. Initially we implicitly assume $d'(\ell', r') = 1$ for all the pairs. We place the tuple $\langle \ell, \mu(\ell), d_\mu(\ell, \mu(\ell)) \rangle$ on a heap.

We then repeat the following until the heap is empty. Let $(\ell', r', \cdot)$ be the pair that has the smallest distance on the heap. We set $\mu(\ell') = r'$. Now, we go through all the $\ell'' \in L' \setminus L$ that co-occur with ℓ' and all the $r'' \in R' \setminus R$ that co-occur with r' and update the estimated distance $d'(\ell'', r'')$ using the new mapping information as

$$d'(\ell'', r'') - \min\left(\text{cooc}'_N(\ell'', \ell', Q_A), \left|\text{cooc}'_N(\ell'', \ell', Q_A) - \text{cooc}'_N(r'', r', Q_R)\right|\right).$$

If the $\langle \ell'', r'', \cdot \rangle$ exists in the heap, we update its distance by $d'(\ell'', r'')$; otherwise, we insert the $\langle \ell'', r'', d'(\ell'', r'') \rangle$ into the heap.

3.4.4 *Using an inverted index*

In this section we propose a way to speed up the computations by using a reverse index. We compute an index I on the tokens in R such that $I(r)$ will return all the tokens that co-occur with r. Now, given current mapping μ and an $\ell \in L$, we can quickly compute the distance to any $r \in R$ by using the following set:

$$S_\ell = \bigcup_{\substack{\ell' \in L| \\ \text{cooc}(\ell, \ell', Q_A) > 0}} I(\mu(\ell')).$$

If $|S_\ell| \ll |R|$, then we gain. Note however that if ℓ' is a high-frequent hash, then $\mu(\ell')$ is a corresponding high-frequent token and so $|S_\ell|$ could be large, rendering this whole method less attractive.

4. ANALYSIS

In this section we describe larger-scale experiments on our base query logs, then turn to an evaluation of the impact of varying the size of the query logs, and the distance in time between the capture of the raw and anonymized log files. In the following section, we move to a discussion of particular privacy breaches that are possible under token-based hashing.

4.1 Larger-scale matching experiments

In this section we employ matching algorithms that successively matches the most frequent 1K, 2K, 4K, 8K, and 16K tokens and hashes in the log-file. The technique is vocabulary expansion with a single greedy update at each expansion stage. Table 2 shows basic data to characterize the information available to the mapping algorithm at these scales. As the table shows, the 1000-th most frequent term appears around 1000 times; and for a sub-graph consisting of only the 1000 most frequent terms, the average degree is about 300 (i.e., on average each term co-occurs with 300 other terms in the top 1000). As we move to 16K terms, the frequency of the least frequent term is 52 in the hashes and 63 in the tokens, so the total available information becomes sparser.

The results are shown in Figure 4, which shows for each depth the performance on the entire range, plus the performance on the top and bottom half of the range. Table 3 gives the actual accuracy at each expansion increment. We are able to perform the inversion with accuracy 99.5% for the first 1K hashes, dropping to 68% for all 16K hashes. To give some insight into these results, it is possible to ask how many hashes, if the mapping μ of all their co-occurring terms were perfect, would in fact have lowest distance to their correct match—this may be seen as a difficult threshold for an algorithm to beat. This is about 92% for 10K terms, compared with 83% for our algorithm at 8K terms, indicating that while there are still gains to be had, the matching is becoming quite difficult as the tail gets longer.

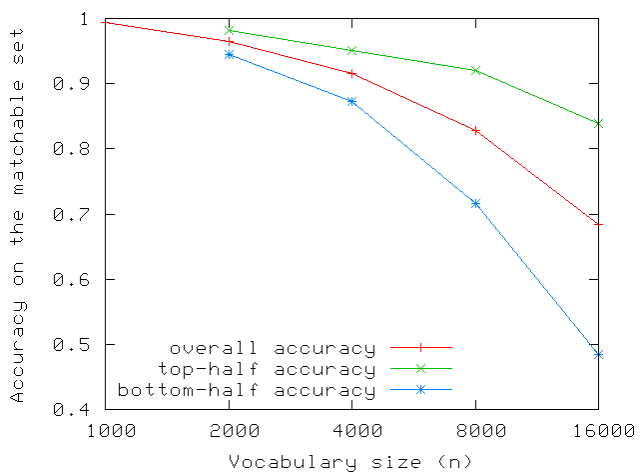

Figure 4: Accuracy of expanding the vocabulary with distance approximations.

n	1K	2K	4K	8K	16K
Accuracy	0.99	0.96	0.92	0.83	0.68

Table 3: Accuracy for matching up to 16K terms/hashes.

4.2 Varying the query logs

We now turn to an examination of how variation in the raw and anonymized logs impacts the performance of the algorithms.

First a note on terminology. For a query log Q, we use *interval* to denote the time difference between the start and end time when the query log was collected. For a raw query log and an anonymized query log, we use *gap* to denote the time difference between the start time of the anonymized log and the start time of the raw log.

For some of the experiments presented in this section, we seeded the algorithm with a "bootstrap mapping" consisting of a small number of correct mappings in order to allow faster convergence; however, this mapping did not have a significant impact on overall accuracy.

4.2.1 *Effect of the query log gap*

Recall that gap refers to the time between the raw query log and the anonymized query log. We took a random sample of 3 million queries from a six-hour interval of time for both the raw and anonymized query logs. For efficiency, we use the heap-based continuous update method, with an initial bootstrap mapping of 100. The vocabulary size was set to 1000. We show results for matching 1K terms, for various values of the gap. The results appear in Table 4, which shows non-monotonic behavior: we perform very well with a gap of one week compared to a gap of one day. This might reflect some weekly periodicity in the query logs.

Gap	1dy	1wk	1mo	2mo
Accuracy	0.74	0.95	0.70	0.77
Matchable	930	915	853	892

Table 4: Accuracy with different gaps between the tokens (R) and the hashes (L).

4.2.2 *Effect of the query log interval*

The goal of this experiment is to measure the impact of the interval of a query log on accuracy; recall that by interval we mean the start and end times of the query logs. We use the raw query log data starting May 17, 2006 and the anonymized query log data starting July 17, 2006. We considered intervals of one hour, three hours, six hours, nine hours, one day, and one week intervals. For each interval, we took a random sample of 3 million queries from the raw and anonymized query logs. For efficiency, we use the heap-based continuous update method, with an initial bootstrap mapping of 100. The vocabulary size was set to 1000.

The results are shown in Table 4.2.2. The matchable set is quite high for different interval sizes implying a large overlap in the queries, irrespective of the interval size.

	token side statistics		hash side statistics	
number of queries	3,849,916		3,187,228	
vocabulary size (n)	freq. of the least freq. term	average degree in graph	freq. of the least freq. term	average degree in graph
1000	1406	333.8	1181	303.4
2000	737	366.3	598	326.9
4000	358	334.9	296	294.1
8000	159	266.0	131	231.1
16000	63	187.5	52	160.7

Table 2: Basic statistics of the data. Dataset for vocabulary size n consists of the subsets of the query logs with only top-n (i.e., n most frequent) terms.

Interval	1hr	3hr	6hr	9hr	12hr	1dy	1wk
Accuracy	0.91	0.95	0.82	0.97	0.96	0.98	0.98
Matchable	874	844	915	892	883	894	899

Table 5: Accuracy for different intervals between the start and end times for each query log.

5. DANGEROUS LIAISONS

In this section we perform an analysis of the breaches in privacy that may be revealed by the level of hash inversions we have shown to be possible in Section 3. First we define key categories of entities that (arguably) reveal privacy. Next we consider the portion of the query log where the hash inversion makes almost no mistakes, and study occurrences of these privacy-relevant entities.

5.1 Privacy-relevant entities

We selected key categories of privacy-relevant entity types that we spot in query strings: person names, company names, place names, adult terms, and revealing terms. We define these five categories below, and describe how we performed the extraction.

i. Person names.

We built a simple context-free name spotter suitable for use in short snippets of text based on "dictionary" lookup constrained by a number of hand-crafted rules. To begin with, we formed a set of potential names by pairing up all firstnames and lastnames from the top 100K names published by the US census. For a firstname-lastname pair to be considered valid, it must satisfy at least one of the following three conditions:

(1) The firstname-lastname pair is present in a list of manually maintained true names.

(2) Either the firstname or the lastname is absent from a small English word dictionary.

(3) The frequency of either the firstname or the lastname in the census data is less than 0.006.

In addition, the firstname-lastname pair must not be present in a manually maintained list of false names. We performed manual evaluation over random samples to determine which query strings actually correspond to names to verify that names identified in query strings that satisfy the above conditions are indeed valid person names.

Not surprisingly, most occurrences of person names in query logs are famous people of one flavor or another. We define a subset of person names to be *non-star* names. These are names that occur fewer than 10 times in the log; we chose the threshold by hand based on the point at which few famous names appeared.

ii. Company names.

We employed the Forbes top 1000 public companies, and top 300 private companies, plus a number of abbreviations added by hand. We perform case-insensitive matching of these company names to the log. Any queries ending in "inc" or "corp" are also tagged as relevant to companies.

iii. Places.

We gathered the names of all US states, and their abbreviations (with the exception of OR). A word followed by a US state or followed by "city" or "county" is considered to be a place name if it occurs in the dictionary capitalized, or doesn't occur in the dictionary.

iv. Adult terms.

These are gathered by scanning the top 2K most popular terms in the query log, and manually annotating those that are clearly adult searches. We selected 14 adult terms. In a log of 3.51M queries, adult terms occur 71418 times, covering about 2% of all queries.

v. Revealing terms.

These are terms that are not adult *per se*, but nonetheless have implications for privacy, if they were revealed for instance to an employer or a spouse. We selected 12 revealing words for this study: `career`, `surgery`, `cheats`, `lesbian`, `disease`, `hospital`, `jobs`, `pregnancy`, `medical`, `cheat`, `gay`, and `cancer`. In a log of 3.51M queries, revealing terms occur 37593 times, covering about 1% of all queries.

5.2 Analysis

Our previous analysis, using query logs without session information, showed that the first 1K most frequent terms of a query log can be mapped with accuracy over 99% on the matchable set. We assume this carries over to a different query log with session information. In the top 1K most frequent terms in this query log, we find 1839 person names, 948 places, and 82 companies. By analyzing co-occurrences within a session, we find the following within-session results. The first column in Table 6 gives the number of sessions that contain an entity or a combination of entities specified in the second column. From the table, it is evident that even the

top 1K terms of the query log contains potentially privacy-relevant information.

Session count	Entity type
7417	unique person name
83801	unique company name
7769	unique place name
2960	unique non-star name
83	non-star name and a place name
169	non-star name and a company name
12	non-star name and an adult term
14	non-star name and a revealing term

Table 6: Number of sessions with privacy-relevant entities in top 1K terms of the query log with session information.

If the anonymized log-file does not include session information, there are still potentially privacy-revealing queries. Expanding to 8K mapped terms, using the mapping achieved by our algorithm, we find within correctly mapped individual queries the following occurrences of potential privacy breaches. Table 7 shows the counts for queries with unique occurrences of the entities.

Query count	Entity type
4816	name
2072	place
220	company
84	name and place
9	non-star name and place

Table 7: Number of queries with privacy-relevant entities in top 8K terms of the query log without session information.

5.3 Mismatches

Finally, we found a number of mistakes made by our algorithm, which give insights into the difficult cases, as well as the types of co-occurrences that are common in query logs. Some example mismatches are shown below. Not surprisingly, since we seek to map hashes into words with similar contexts, quite a number of hashes are (reasonably) mapped into synonyms or paraphrases of the original tokens, as well as related concepts that tend to appear in similar contexts. Since these types of mismatches are semantically equivalent or related to the correct matches, they may still be very effective in incurring privacy breaches. Not all mismatches remain "helpful" in this way. With limited amount of data and noise incurred by the non-overlapping part of the vocabulary, some of the hash-token pairs may never get correctly mapped and remain as misleading contexts for other pairs. Thus, it is not surprising that we also have inexplicable mismatches where there are no obvious semantic relations between the two words.

Synonyms.
retreat↔getaway
furnace↔ fireplace
pill↔ supplement
pics ↔ photos

Terms used in similar contexts.
celine↔ elvis
may↔ april
positive↔ negative
heel↔ toe
pilates↔ abdominal
wants ↔ millionaire
avis ↔ hertz

Unexplained.
killer ↔ crack
origami ↔ biodiesel
suicide ↔ geometry

6. CONCLUSIONS

In this paper we studied the natural token-based hashing in which each search string is tokenized, and each token is securely hashed into an identifier to create an anonymous query log. We show that serious leaks are possible whether the identifiers are presented in the same order as the underlying tokens, or whether the order is hidden. We thus show that user concerns around privacy are very real at least in the case of token-based hashing.

Future work includes expanding the scope and applicability of our algorithms to make them work for large values of n.

7. REFERENCES

[1] R. Barzilay and K. McKeown. Extracting paraphrases from a parallel corpus. In *Proc. of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 50–57, 2001.

[2] R. J. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *Proc. of the 21st International Conference on Data Engineering*, pages 217–228, 2005.

[3] A. Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, 2002.

[4] B. J. Jansen, A. Spink, J. Bateman, and T. Saracevic. Real life information retrieval: A study of user queries on the web. *SIGIR Forum*, 32(1):5–17, 1998.

[5] B. J. Jansen, A. Spink, and T. Saracevic. Real life, real users, and real needs: A study and analysis of user queries on the web. *Information Processing and Management*, 36(2):207–227, 2000.

[6] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *Proc. of the 15th International Conference on World Wide Web*, pages 387–396, 2006.

[7] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison Wesley, 2005.

[8] L. Lee. Measures of distributional similarity. In *Proc. of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 25–32, 1999.

[9] R. Lempel and S. Moran. Optimizing result prefetching in web search engines with segmented indices. *ACM Transactions on Internet Technology*, 4(1):31–59, 2004.

[10] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *Proc. of the 23rd ACM Symposium on the Principles of Database Systems*, pages 223–228, 2004.

[11] J. Novak, P. Raghavan, and A. Tomkins. Anti-aliasing on the web. In *Proc. of the 13th International Conference on World Wide Web*, pages 30–39, 2004.

[12] R. Pang and V. Paxson. A high-level programming environment for packet trace anonymization and transformation. In *Proc. of the ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 339–351, 2003.

[13] F. Pereira, N. Tishby, and L. Lee. Distributional clustering of English words. In *Proc. of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 183–190, 1993.

[14] D. E. Rose and D. Levinson. Understanding user goals in web search. In *Proc. of the 13th International Conference on World Wide Web*, pages 13–19, 2004.

[15] N. C. M. Ross. End user searching on the internet: An analysis of term pair topics submitted to the excite search engine. *Journal of American Society of Information Sciences*, 51(10):949–958, 2000.

[16] P. Samarati and L. Sweeney. Generalizing data to provide anonymity when disclosing information. In *Proc. of the 17th ACM Symposium on the Principles of Database Systems*, page 188, 1998.

[17] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.

[18] A. Slagell and W. Yurcik. Sharing computer network logs for security and privacy: A motivation for new methodologies of anonymization. In *Workshop of the 1st International Conference on Security and Privacy for Emerging Areas in Communication Networks*, pages 80–89, 2005.

[19] A. Spink. A user-centered approach to evaluating human interaction with web search engines: An exploratory study. *Information Processing and Management*, 38(3):401–426, 2002.

[20] A. Spink, B. J. Jansen, D. Wolfram, and T. Saracevic. From e-sex to e-commerce: Web search changes. *Computer*, 35(3):107–109, 2002.

[21] A. Spink and H. C. Ozmultu. Characteristics of question format web queries: An exploratory study. *Information Processing and Management*, 38(4):453–471, 2002.

[22] S. Zhong, Z. Yang, and R. N. Wright. Privacy-enhancing k-anonymization of customer data. In *Proc. of the 24th ACM Symposium on the Principles of Database Systems*, pages 139–147, 2005.