User 4XXXXX9: Anonymizing Query Logs

Eytan Adar
University of Washington,
Computer Science and Engineering
eadar@cs.washington.edu

ABSTRACT

The recent release of the American Online (AOL) Query Logs highlighted the remarkable amount of private and identifying information that users are willing to reveal to a search engine. The release of these types of log files therefore represents a significant liability and compromise of user privacy. However, without such data the academic community greatly suffers in their ability to conduct research on real search engines. This paper proposes two specific solutions (rather than an overly general framework) that attempts to balance the needs of certain types of research while individual privacy. The first solution, based on a threshold cryptography system, eliminates highly identifying queries, in real time, without preserving history or statistics about previous behavior. The second solution attempts to deal with sets of queries, that when taken in aggregate, are overly identifying. Both are novel and represent additional options for data anonymization.

1. INTRODUCTION

In August of 2006, American Online (AOL), in a move designed to help researchers in the Information Retrieval (IR) community, provided an extremely large query log extracted over 3 months from their search engine [3]. Though the release of query logs is nothing new, the sheer size of this collection and the lack of any significant anonymization was unprecedented. The data represented ~650k users issuing 20 million queries which were recorded in the "clear." The result of this release was the disclosure of private information for a number of AOL users, major damage to AOL [1], and a significant damage to the research efforts of academics who depend on such data.

The most troubling aspect of the data was the ease by which single unique individuals could be pinpointed in the logs. Even ignoring the existence of social security numbers, drive license numbers, and credit card numbers, the New York Times (NYT) [5] demonstrated the ability to determine the identity of a real user. In this case, the user corresponding to ID 4417749 was found to be Thelma Arnold, a 62 year old woman living in Georgia. Ms. Arnold narrowed down her location by making multiple queries for businesses and services in Lilburn, GA. While Lilburn, GA has over 11k citizens, Ms. Arnold further helped the NYT by making a number of queries for a Jarrett Arnold (and other members of the Arnold clan). From here, the NYT could contact all citizens of Lilburn with the last name Arnold (of which there are only 14 according to the Yahoo whitepages). Though it is difficult to quantify the exact number of users that could be identified in this way, the mere possibility has led to fear in the user community.

Already, various class actions suits and complaints have been leveraged against AOL[2][4]. Some have demanded that search logs not be retained at all by the service providers. For various reasons, this is likely not practical for search engine vendors. The need to improve their service both for users and advertisers requires certain information be kept. Furthermore, recent trends in personalized search illustrate the practical application of prior knowledge about users.

This work is based on the following assumptions: a) user search logs are retained for at least some period by the search vendor, b) those logs may be disclosed or leaked, and c) the vendor would like to ensure that whatever data is leaked or released does not have significantly identifying information. Additionally, we would like to recognize that there is a tradeoff between the potential usefulness of a log and how much anonymization it undergoes. For our purposes, identifying information is the set of queries that a user issues that can be used to identify the individual uniquely or at the very least reduce the search space to a manageable level. Note that we are not explicitly concerned with recovering traces for law-enforcement or other officials and do not consider this a design goal. However, the search vendor may retain information that could make this possible (at a potential privacy risk to their users).

The results presented here illustrate how by simply removing infrequent queries, identifying queries are also removed. We also describe a cryptographic technique based on secret shares that allows the search vendor to "set it and forget it." By applying this technique to the data, a person who has obtained the log is constrained to reading only those queries that have been used by at least t other individuals (where t can be fixed by the log creator). While masking infrequent queries goes a long way to mask identity, we must also worry about the combination of more common facts can be considering identifying. Obviously, the longer the search history for a user, the more potential there is for finding these kinds of intersections. To eliminate this form of attack, we propose a number of mechanisms for breaking apart sessions into less identifiable chunks.

We briefly consider the "power" of data sanitized in this way relative to the needs of the data miner.

1.1 The Privacy/Utility Tradeoff

The Privacy-Utility tradeoff is based on the understanding that the more data we eliminate from a log through anonymization, the more privacy we convey to users but the less that data is useful to those seeking to mine the data. The difficulty in making a precise claim about how much privacy or how much utility we are trading off has much to do with the imprecision in defining the two concepts themselves.

While it may be more clear-cut that a search engine should not publicly disclose credit-card numbers that appear in their logs, there is a sizeable gray area. For example, some users may consider it OK for the search engine to have access to the queries for personalization purposes but not for advertising. At one

extreme, some users would like none of their searches logged and consider it a violation for any information retained, let alone revealed to third parties. At the other extreme, there are users who are happy, for whatever reason, to share their query history. However, as publishers and consumers of logs we must consider the fact that some users do not understand the privacy implications of certain queries or sets of queries, which when revealed, would cause them harm. Finally, we must also acknowledge the impact of the adversarial model in this analysis. Asking who is the adversary "violating" privacy safeguards, in what ways they would do it, and what their capabilities are, is an art in itself and may not be something the community is capable of doing correctly.

Utility is generally analyzed with a specific task in mind. If we are simply seeking to measure the time between queries, we do not require any identifying information from the query logs. However, the more complex the task the more privacy we must take away from users (e.g. infering the demographic characteristics such as zip-code and salary based on the query terms a user searches for). Furthermore, we must understand what inferences can be made and with what confidence on any log data. For example, while a user's gender may be known by the search engine, the log may not contain this information. However, some inference of gender may be possible based on search behavior and key phrases and while absolute certainty will not be possible, the results will still have some high accuracy (e.g. [12]).

In this work we introduce two point solutions in the space of possible anonymization techniques. We have selected a specific privacy objective and a specific utility objective. In particular, we have attempted to remove information which may be used to uniquely identify a user while still providing un-encrypted access to a portion of the search logs that may be sufficient for certain personalization tasks. While we can not perform linguistic analysis on *every* query with the mechanisms we describe or extensively analyze the so-called "long tail," we still know the distribution of queries for a user, query timing, and also the content of many queries. Furthermore, it should still be possible to cluster users and to some extent augment search engine responses with user behavior. More significantly, if only the first of our techniques is used (that of masking unique queries) we can also correlate queries (e.g. those who query for X also query for Y). This is possible even without knowing the specific terms that the hashed queries that X and Y represent.

This type of correlation is, unfortunately, not possible when masking users that have unique sets of queries as we propose in our second technique. The more aggressive the masking, the fewer the correlations we can draw. This anonymization is a move in the privacy direction and potentially away from utility. However, by clearly defining the types and quantity of information that is masked using the techniques we describe, we believe that both users and miners will have a clearer understanding of potential privacy and utility implications.

2. RELATED WORK

Although there is not a great deal of work on anonymizing search logs, it is also worth considering how others have approached this problem and the various tradeoffs they have made.

The bulk of previous work in data anonymization has been concentrated in two distinct communities. The database community has applied techniques such as statistical databases [8] and k-anonymity [22] to attempt to provide certain levels of measurable privacy (a more complete list of papers is available at [17]). The difficulty with applying such approaches is they are frequently costly (providing non-approximated k-anonymity is NP-hard) and are not intended to deal with data that is changing rapidly. Search logs are produced at a very fast rate and are likely to require a different mechanism. This makes the anonymization techniques from the database literature, which are primarily intended to work on static data sources, difficult to apply.

The other major research thrust on anonymization has been in the network community ([7][14][20][21]). Here, the design objective has traditionally been pseudonymity. Generally, the belief is that most behavior, such as standard network traffic, should be anonymized but that aberrant/illegal behavior should be detected, tracked, and reported. Thus, most pseudonymity schemes involve mechanisms for recovering the masked information, frequently through third-party or communal agreement. It is also worth noting that work in this area has generated server based solutions that allow searching on encrypted log files [24] as well as solutions whereby log files are retained at the server and analysis code, which only emits aggregate statistics, is shipped to the client [14]. While laudable in that it addresses the needs of third-party data miners, a general approach of this type is not currently available, and requires so many safeguards that it may not be practical without restricting users to the most basic operations.

The most similar system to our own is [9]. The system proposes to anonymize Unix log files by a secret sharing scheme. Interestingly, however, the system is designed with nearly an opposite motivation to our own. While we have tried to reveal log entries that are common to many users and mask those that are unique or "aberrant," the goal of [9] is to expose those users while keeping the average user hidden.

In the context of anonymized search logs, we are aware of a single recent study ([10]) that demonstrates how token based hashing is ineffective for log anonymization. The work illustrates that a statistical attack on hashed tokens may lead to privacy violations. We avoid this problem by explicitly not hashing tokens but rather encrypting entire queries, making decoding attacks much more challenging.

3. DATASET PROPERTIES

The AOL dataset consists of 10 files containing nearly 37M lines of data representing 657k users over 3 months (March 1, 2006 to May 31, 2006). Though AOL is unique in that users log into the system and are more readily tracked we note that other search engines achieve similar tracking through the use of cookies and features that incentivize users to log in (e.g. email, instant messaging, etc.). Full descriptive characteristics of the data are available in [15]. The data is of the format:

{*AnonID, Query, QueryTime, ItemRank, ClickURL*}

Where *AnonID* is the anonymous user ID, *Query* is the query string, and *QueryTime* is the time of the query. If the user clicked on a result, this is recorded in the next two fields. *ItemRank* is the rank of the clicked result in the search list and *ClickURL* is, in theory, the URL of the clicked result truncated to the domain name (e.g. www.mit.edu/alumni/ becomes www.mit.edu). This one minor attempt at anonymization appears more targeted at potential business implications (search engine optimizers would like to know which pages get clicked on for which results). However, as we will describe below, even this redaction failed for a number of reasons.

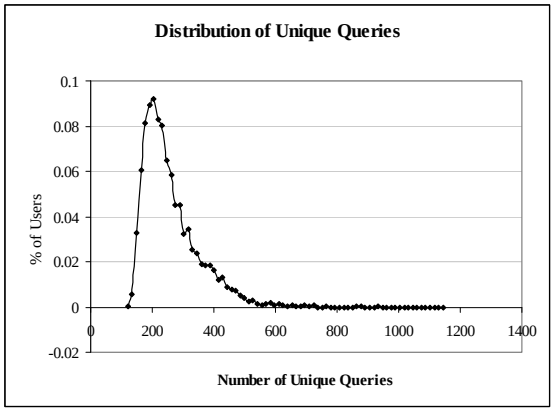

Figure 1: Unique Query Histogram

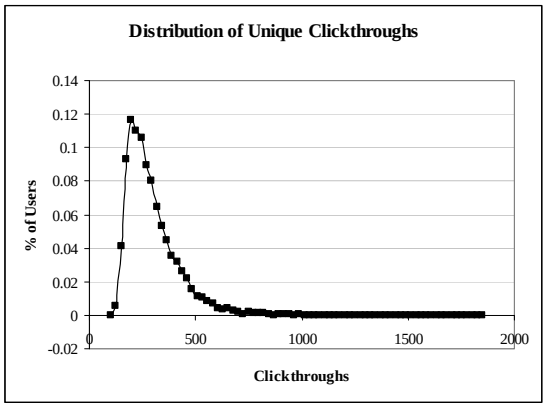

Figure 2: Unique Clickthrough Histogram

For some of the analysis below we will made use of the entire search population. However, to experiment with a number of techniques we collected a sample of users that made between 100 and 300 queries with clickthroughs (5546 users sampled at random). While this limit is somewhat arbitrary, it captures users that make about 1 query a day while eliminating robots or other extreme search behavior. From the time of their first query to the time of their last, users were logged for an average of 87.8 days, with a minimum of 12.33 days and a max of 91.9 days (median 90.1). The average user in this dataset had 571 log entries (494 median), representing 252 unique queries (229 median) and 252 unique clickthroughs (252 median). The distribution of these unique queries and unique clickthroughs is represented in Figures 1 and 2 respectively.

4. IDENTIFYING INFORMATION

It is reasonable to start by asking what kind of identifying information exists in query logs. There are easily 651 numbers that conform to the format of social security numbers. While some of these could be mistyped phone numbers or other queries, a number of them have the words "social" also in the query (22 unique instances). Using a credit card checksum algorithm we find at least 92 numbers that satisfy the format of a credit card and have a valid checksum (whether they are or ever were valid credit card numbers is unknown). These are easily the most egregious of the privacy problems in the logs and may have a significant financial implication to their owners. In contrast to the AOL dataset, we note that in anticipation of exactly this problem, the MSN release of a similar log trace eliminated some of these issues by removing all long numerical sequences from the logs.

Credit card, social security, or driver license numbers are likely only known to their owner—or a small group of people with some connection to the owner—and are thus useful in associating all other queries with an individual users. However, there are other pieces of information that can also be identifying. Queries for phone numbers, addresses, and names of individuals are all useful in narrowing down the population. Even if the names or addresses do not belong to the user themselves, one could conceivably approach the target of the query for further information.

Given the massive privacy damage in releasing credit card numbers and social security numbers, it is easy to overlook other mistakes that AOL made in generating the dataset. Even in their attempt to redact certain information such as full URLs, they mistakenly left the complete URLs to all secure web servers (i.e. HTTPS protocol URLs). There are 84k such instances in the logs. While these are public—the search engine did index them after all—they may in fact be sensitive. For example, it is possible to determine certain items that were searched for on shopping sites. These may be revealing, embarrassing, or illegal and hence endanger the user in some way.

Finally, while redaction of URLs adds some challenges to our analysis, it is fairly easy to recover the full URLs. This is achieved by replaying the queries in the search logs against the search engine, and comparing the results to the domains of the clicked on URLs. AOL even assists us in this by noting the rank of the clicked-on document. With the smaller user set of 5546 users we were able to recover 1.4 million (over 70%) of the URLs that were clicked on. This was done by finding the top results by replaying the queries against the Google and MSN search engines (20 and 50 top results respectively). Though we can not guarantee the accuracy of this recovery absolutely—as the search engines may return new results from the same domain—it is likely that the closer this recovery is done to the time of the log disclosure, the more accurate it is. Additionally, more intelligent searching could increase this recovery number further (i.e. obtaining more results, isolating searches to certain domains, etc).

We do note, however, that we were able to perform this analysis because Google and MSN both provided us with extended querying ability for research purposes, on the order of 100 times the amount allocated to a regular developer. Even with this boosted query limit, reverse lookup of the queries from the 5k users still took a week. Regardless, it is likely that with enough ambition and resources someone else would be able to repeat this step.

5. MASKING IDENTIFYING DATA

One of the simplest things one could do to improve the anonymity of the data is eliminate those queries that are known to be uniquely identifying. While we would prefer to only remove queries that are truly identifying (rather than some very infrequent query), any inference we make will inevitably lead to removal of harmless data. We could imagine, for example, eliminating only those queries that are generated by a small set of users. Eliminated queries could simply be discarded or hashed with an appropriate salt. We are relatively unconcerned about cryptanalysis techniques as the hashed queries are generally "in the noise." That is, there are so many queries that only happen once or twice, it would be impossible or very difficult to determine which one a particular hashed value represents (though as demonstrated by [10], token based hashing is not immune to such analysis). Even making use of non-hashed queries in a user's session to narrow down the possible values of the hashed ones does not help the adversary. An adversary would need to have a significant un-hashed log sample to build a sufficient

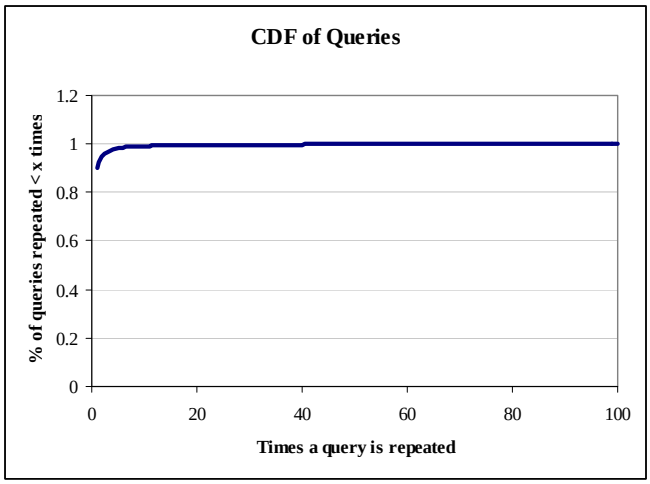

Figure 3: Uniqueness of Queries

probability distribution. Or stated another way, the adversary would need to have seen the actual query before, in the un-hashed sample, to assign it to the hashed one. For example, if we hash a credit card number that uniquely appears in the dataset, the adversary would need to have previously seen the number to reverse the hash, no matter how much other data they have.

The distribution of the data is such that most queries (90%) only appear once. Figure 3 represents the Cumulative Distribution Function (CDF) of queries. We see that the bulk of all unique queries (90% or 9,138,762) are only issued once. In fact, 97% of all queries are issued 3 times or fewer. However, this is not to say that we have eliminated 97% of the logs be eliminating these queries. Because there are nearly 17M queries in the log, eliminating single instance queries leaves us with 46% of the logs. Eliminating 3-instance (or fewer) queries leaves us with 31% of the logs. Furthermore, for this analysis, we apply a fairly conservative definition of equality for counting the number of repeated queries. That is, the two queries must be exactly equal (though the logs themselves are lower-case normalized). There are other measures for equality that attempt to normalize queries [23] (e.g. stemming the word, sort words, etc.) and may not hide as much data.

While masking a large percentage of the queries may not make certain analysis impossible, it is important to recognize that there is some loss in the utility of the data. For example, it is still possible to cluster users based on queries, but difficult to understand how users differ in the specific phrasing of their queries within these clusters.

5.1 Anecdotal Evidence of Effectiveness

While it is impossible for us to say with absolute certainty that there is no single query that is uniquely identifying to some individual we do see some anecdotal evidence that this is likely. For example, eliminating single-instance queries masks all credit card and social security numbers.

In the case of our Ms. Arnold, 224 queries shrink to 84 different queries that were uniquely issued by her. We no longer see her queries for other Arnolds. In fact, we no longer see her queries for her home town. The best we can probably do is to see her searches for the Gwinnett Humane Society. The Human Society serves Gwinnett County, GA, a county with 588k residents. After eliminating 2-instance queries even this clue is gone.

5.2 On-The-Fly Elimination, Secret Sharing

There are two primary issues with simply removing those queries that appear infrequently. The first is the obvious issue that many results that are not truly identifying are eliminated. As more and more data is produced, we may discover that a specific query is in fact not as uncommon as originally presumed, and it would be difficult or impossible to recover data that has been removed. This is especially true if we have a streaming scheme whereby the log provider streams the log data to us and throws out those results it believes are identifying. This belief is based on some training experience but means that certain queries—those that are emerging (i.e. there's a new topic that is being queried about that we hadn't seen before)—are not available to the log analyst. Even if the log provider relearns that a specific query is not identifying, we will have potentially lost the first few instances of those queries. To avoid this, the logging system can buffer the queries for some period and anonymized them in less than real time. However, this means that the logs are being held as unencrypted data for some period which may be undesirable for certain scenarios.

The second issue is that the log provider must keep track of histogram of previously asked queries. For a search provider with any significant amount of traffic this is a potentially expensive proposition.

To solve both issues we propose an application of a secret sharing, or threshold cryptography, scheme [12]. In a secret sharing scheme a secret *S*, in our case a query, is split into a number of shares. Each share is useless on its own, but in combination the secret can be decoded. One version of this scheme is a unanimous consent scheme using modular addition. In this mechanism a secret is split into *t* shares and *t* shares are required to decode the secret. The scheme works essentially by generating *t* -1 random numbers ($S_1 \ldots S_{t-1}$) in the range of 0 and *m* – 1. A final secret, S_t, is generated as:

$$S_t = S - \sum_{i=1}^{t-1} S_i \bmod m$$

For our scheme we would like to make sure that a query must appear *t* times before we can decode it. To achieve this we can create *t* hash buckets and map the user ID to one of these buckets (e.g. *UserID* mod *t*-1). The bucket indicates which secret share is to be used, and the query is replaced with the appropriate share and a query ID. So that we do not need to remember all the previously generated shares for a given query we can seed the random number generator deterministically, for example, using $H(P_k, <query, BucketID=UserID \bmod t - 1>)$ (where H is some encryption function and P_k is a private key). Thus we replace the string for any given query, q_i, by user u_j, with:

$$< u_j, q_i > = < u_j, H(P_k\, q_i), S_{ji} >,$$

where S_{ji} is the secret for user *j* for query *i* as described above. The disadvantage of such as scheme is that it is possible that we will initially need to see more than *t* unique users making the query before we are able to decode this (i.e. the first *t* users hash to fewer than *t*-1 buckets). While this is unfortunate, as it is unpredictable, it may not be unreasonable especially with a low threshold of 2 or 3 shares. For example, imagine the exponential decay of the probability of repeatedly getting the same share with $t = 2$ (i.e. $.5^k$, with *k* being the number of users issuing the query).

An alternative scheme is a Threshold Scheme (TS) in which the secret is split up into *n* pieces where any *t* of those pieces can be

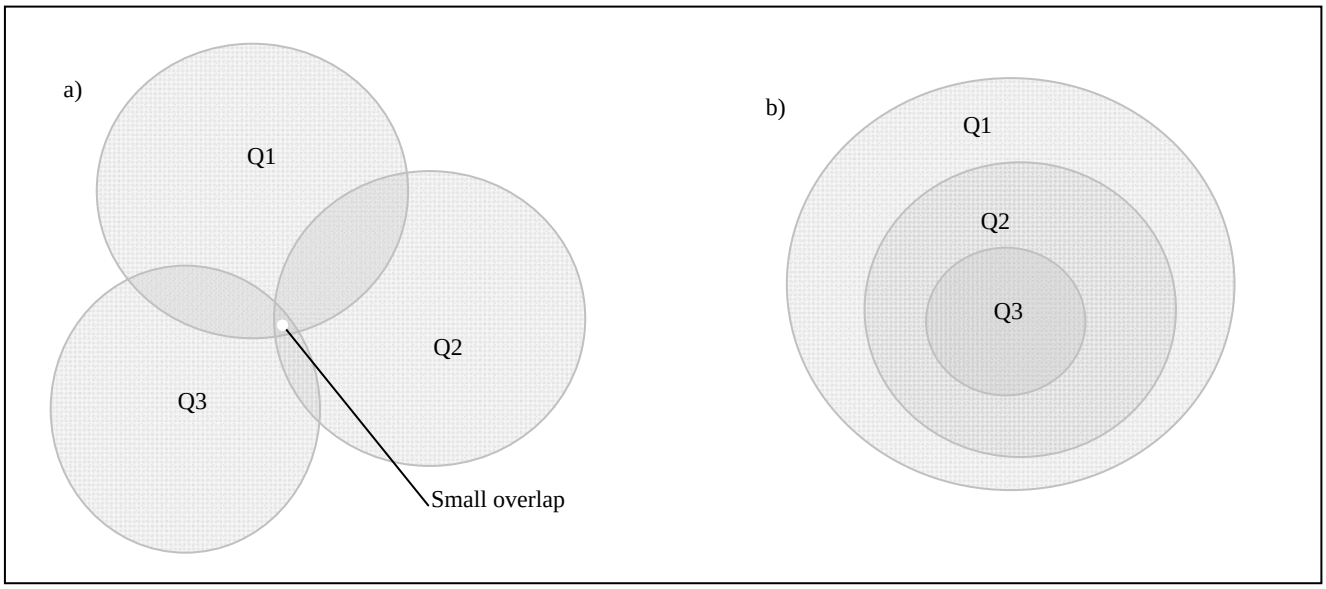

Figure 4: Two scenarios for a given user. In a) the user has queried for {Q1,Q2,Q3}, which have some overlap. The user is uniquely contained in the intersection. In b) we choose {Q1,Q2,Q3} to ensure a nesting relationship so that the user is as identifiable as the most unique query.

combined to decode this secret. By choosing a sufficiently large n we can ensure that the probability of getting any t distinct secrets given k choices is high. The setup for such a scheme is as follows ([12][18]):

- Select a prime $p > \max(S, n)$ and let $a_0 = S$
- Select $t - 1$ random coefficients, $a_1 \ldots a_{t-1}$, where $0 \le a_i \le p - 1$, defining the polynomial:

$$f(x) = \sum_{j=0}^{t-1} a_j x^j$$

- Compute $S_i = f(i) \bmod p$, $1 \le i \le n$. The pair $< i, S_i >$ is the share

To pool the shares, t distinct shares are collected, and are used to compute the coefficients by Lagrange interpolation.

5.3 Performance

There are a number of freely implemented secret sharing schemes such as SSSS[1] and JSS[2] (based on the algorithms described in [19]). In order to test the feasibility of this method of on-the-fly encryption we modified the JSS system to use deterministic randomization (as described above) and processed 1M lines from the AOL query log (ordered by time). The difference between the first and last query in this trace is ~18000 minutes or 12 days. In the experiment we opted for using 10 shares, with $t = 3$ (i.e. 3 distinct copies required to decode). Because the algorithm creates all shares the first time the secret is encrypted, we cached the results using a soft-pointer scheme that removed infrequently used shares during garbage collection.

Once the cache was warmed up (within reading ~2000 lines from the log) we find that each log entry consistently requires 50ms (on average) to encrypt on a Dual-Processor, Dual-Core (3.2Ghz) system. The total encoding time was 821 minutes or approximately 13 hours. Note that this time represents an upper bound on performance as many improvements can be made to the implementation (i.e. parallelization and pre-generating or caching a large set of primes would speed things up greatly).

Decoding is done by sorting the encrypted log file based on the query ID and finding sequences in which there are 3 or more distinct shares. Once sorted, decoding is virtually instantaneous (under a minute for the full file).

5.4 Attacks

Attacks on this type of anonymization mechanism are rather limited. In order to decode the hashed query an adversary would need to have the system generate a sufficient number of shares for a specific query. That is, the adversary would need to, a priori, know the text of the query. This would make finding things like credit card numbers impossible, as the adversary would already need to know the number.

While decoding specific messages is not possible, an attacker may attempt to track a specific user. For example, if the attacker knows that a user makes a query (or forces the user to make that query), they can effectively mark the session. Upon obtaining the logs, the adversary would find the uniquely identifying marker and associate the session with a user. It is notable that this attack is possible regardless of the way in which we hash. In fact the hashing scheme requires the adversary to work harder as they need to force the search provider to generate enough shares to find the marked session.

It may also be necessary to pad queries with additional information in order to retain consistently sized log lines. Since share size or encrypted data (depending on the scheme) is proportional in size to the original query certain information may be leaked.

Finally, it is possible to change the private key, P_k, with some regularity in order to limit the size of logs that can be decoded or in the event that a previous private key is obtained in some way.

[1] http://point-at-infinity.org/ssss/

[2] http://www.das.ufsc.br/~neves/jitt/jss.html

6. SPLIT PERSONALITY

While the technique described above goes a long way towards eliminating identifying information, it does so for single queries only. We must also be aware that the combination of queries that are independently common, may in aggregate be highly identifying. For example, there are at least 51 cities/towns named Springfield in the US. Thus a query for "Springfield" is not particularly identifying (there are easily 600k people living in one Springfield or another). Similarly, the query "Louisiana" (which has nearly 6 million residents) is also not particularly identifying. However, a user issuing both queries has potentially narrowed themselves down to 395 residents. So while we may be able to mask the query "Springfield, Louisiana" and protect the user, if they make the two queries separately, they are likely both common enough to escape encryption, and with enough inference power we would be able to greatly reduce the search space. This relationship is illustrated in Figure 4a.

One trivial solution to this problem is to simply forget about user identifiers in the anonymized logs. This unfortunately renders the data nearly useless for many forms of analysis. An alternative is to occasionally "forget" the user's identifier. For example, once a day (every 24 hours) we could assign a completely new ID to the user. Alternatively, we could assume that continuous browser sessions are fairly small and do not cover a wide range of topics. Therefore an alternative mechanism is to approximate a search session and forget the user ID after a period of inactivity (say 30-60 minutes). Both techniques greatly reduce the size of the queries in each "session" and increase the number of sessions. For the 5546 user sample we move from 5546 traces to 250,858 sessions when splitting on a daily basis, and 581,459 when splitting on hour long gaps (session lengths are an average of 12.64 and 5.45 queries respectively). However, as we will demonstrate below, (and consistent with [16]) this is still vulnerable to attack. Certain queries and query patterns are so uniquely identifying that it is possible to recombine some smaller sessions into the original trace.

An alternative solution to this problem relies on the observation that users have multiple "interests." Each interest has a corresponding set of queries that are related to that interest. For example, if a person is interested in football, they may query for "Football," "ESPN Football," and "Seattle Seahawks." If they are interested in cooking they may query for, "Farmer's market, Seattle," "recipe guide," and "recipes with yams." Users tend to have many interests and so their queries become entwined. Glancing at any contiguous section of their query log one may see a range of the users interests. It is this characteristic that allows us to reconstitute search logs that are split in the ways described above. Instead, what we suggest is splitting the user's identity based on their interests. That is, a user with football and cooking as two separate interests will be made to look like two distinct users. Each version of the user shares nothing in common with other versions.

What is important to recognize is that we are attempting to group queries so that we can force the nesting relationship diagrammed in Figure 4b. That is, there is some large query bubble corresponding to the user's general interest (the larger the bubble the less unique the query). Each query within the user's interest is contained within this larger bubble. Therefore, a user is as uniquely identifiable as the most identifying query (which can be limited by the scrubbing techniques described previously). Clearly, this is the idealized model. In reality, we still may have instances of overlapping queries (e.g. "Springfield football" and "Louisiana football"). The objective, however, is to select a clustering criteria that is sufficiently specific to reduce the number of such cases.

For the purposes of this analysis we apply a number of simple techniques to determine the similarity of two queries.

- The similarity between two queries is judged by the similarity of the result sets returned by each query form the search engine. The more similar the results sets, the more related the queries. While we have experimented with more sophisticated notions of similarity (e.g. a TF-IDF scheme [6] that counts search results as "terms") we have found that any overlap in the top 50 results for two queries indicates a sufficient similarity to judge two queries to be related. Though this will clearly change with access to more search results, in our tests, the function resultsim(q_i,q_j) returns a 1 if any results overlap and 0 otherwise.

- A second similarity metric is the Levenshtein distance [11] which is simply a measure of the text edit distance between two strings. This captures minor spelling mistakes (e.g. "helo" and "hello"). We have found that a value of .2 or lower appears to work, though a more rigorous analysis may help refine this.

- A final similarity metric is the word overlap given by the Jaccard metric. Given two query with multiple words, we calculate the Jaccard coefficient by dividing the number of words the two queries have in common, with the number of words in the union of the two queries (e.g. "walk me to the moon" and "walk to moon"). Similarly to above, we have found that a value of .5 or more appears to work.

Using these similarity measures we can now build our profiles. Let Q be an ordered list of queries issued by a user, and q_i be the i^{th} query. Let D be an ordered list of numerical IDs assigned to each query and let d_i be the ID assigned i^{th} query. We then apply a pseudo-hierarchical clustering technique which starts from the first query and works its way down the list. A query that is similar to one we've seen before gets the same ID of that previous query. If a new query is too different it receives a new identifier. The algorithm in pseudo-code corresponds to the following:

```
d_0 ← 0, k = 0
for i = 1… |Q| – 1
  for j = 0… i – 1
    if  q_i = q_j, let d_i ← d_j
    else if resultsim(q_i,q_j) != 0, let d_i ← d_j
    else if levenshtein(q_i, q_j) <= .2, let d_i ← d_j
    else if jaccard(q_i, q_j) >= .5,  let d_i ← d_j
    else k = k + 1, let d_i ← k
```

The output of this process are 1,025,900 sessions (avg. 3.1 queries per session).

In the example of our Ms. Arnold, we now find that she has 165 different personalities. The system correctly recognizes that queries about "retirement in new zealand" and "new zealand" are related. Unfortunately, given our thresholds it does not decide that "retirement in australia" is related. However, the system does know that "applachion trail" and "appalachian trail" are variants of the same query. Hopefully, with some experimental tweaking the system can maintain accuracy while reducing the false negative count.

6.1 Evaluation

From the perspective of utility, this form of pre-processing limits us to one specific facet of an individual. Thus, if a researcher is interested in correlating completely different facets (e.g. a person who queries for "football" also queries for the "food network") they will not be able to use a dataset encoded in this way.

Ideally, when we generate these sessions each session would contain a set of queries that are vastly different from other sessions generated for the same user. To test our different splitting criteria we built a profile search engine. Each session, as generated by the three techniques, was indexed by taking all the query terms used in that session and combining them into one larger text "document." Each document was then indexed in a standard text database. To query the system we would take a given session, transform it into a document as described above, and find the most similar document/sessions in the index. Ideally, the top results returned by the profile search engine would not correspond to any of the user's other sessions.

To test this we built indices for each of the three session types (daily splits, hour gap splits, and interest split). We then took each session and queried it against the generated indices. Obviously, the top result was always the session we submitted. We recorded up to 100 matching sessions for each input session.

To evaluate the "fragmenting" effectiveness of the different split techniques we looked at two things. The first was a count of how many of the top 100 sessions returned for every query corresponded to the original user. That is, if the input session belonged to user 5, and it matched against the (user,session) pairs ((6,1),(5,2),(5,3),(7,10)) we would say that ½ of the top results were from the same user. Ideally, the selectivity ratio of same-user results to all-user results will be 0. If it is 1, that means all similar sessions belong to the same user. Using this metric we can find the average selectivity for the different splitting criteria.

We calculate that, on average, for daily-split sessions, 6.5% of the top results returned for each session as a query belonged to the same user (6 out of 100 of the results belong to the same user). For the hour-gap split we see an average of 9.2% (1 out of 10). As expected, for the profile-split we only have 2.3% in the top list.

One potential way in which an adversary can do better is by only trying to find sessions from the same user when the number of sessions returned is small. That is, if the adversary queries for a session and gets 1M results, that indicates that the query session has some similarity to pretty much all other sessions. On the other hand if there are only 1000, 100, or even 10 responses it is likely that the matches are more accurate. We find that this actually happens in the data. In Table 1, we see that as the number of responses decrease, the number of matches in the top lists increase (up to 2 out of 10 for the daily and hour-gap splits). However, the number of same-user matches remain at under 4% for the profile-split.

Table 1: Reconstruction of split sessions. Number of other sessions by the same user returned by the session search engine depending on split type and the number of responses.

	Any # of responses	**≤ 1000 responses**	**≤ 100 responses**	**≤ 10 responses**
Daily	6.4%	12.1%	20.9%	19.1%
Hour gap	9.2%	15.9%	22.8%	20.7%
Profile	2.3%	2.5%	3.6%	4%

The second mechanism for evaluation is trying to reconstruct the original query log. We do this by assuming an adversary with an oracle. Given an input query, the adversary gets a list of matching sessions. Given two sessions, the oracle returns true if the two sessions come from the same user and false otherwise. Thus, the adversary may go through the returned results picking out the ones that belong to the same user. This represents a worst case scenario, but sets an upper bound on recoverability. To reconstruct the logs we query all sessions for a given user against the database. For every result in the top 100 that comes from the same user we draw an edge between the query session and the matching session. At the end, if all sessions for a given user are connected in a graph we have perfectly recovered the user's log. In reality, what we end up with are a number of connected components. If successful, a good splitting technique will mean that for a given user we have as many connected components as we have sessions.

To calculate the reconstruction ratios we begin by finding the optimal "ratio" for any given user. If a user has been split into k sessions, we have an optimal ratio of $1/k$. Our recovery ratio is the m/k where m is the number of connected components (as determined above). The quantity of *optimal-ratio / recovery-ratio* will approach 1 as we recover more and more sessions. What we find is that for the daily-split this ratio is, on average, 20.6, for the hour-ratio it is 56.2, and for the profile split it is 130. Recalling that, on average, each of the 5546 users was split into 185 sessions, we have not recovered a significant piece of the trace.

The main issue with the approach proposed here is that we are no longer able to claim real-time, memory-free encoding of the logs. While with enough computation power we may be able to create near real-time splits, we would need to keep around a list of topics/interests that the user has previously queries around. A possible fix for this issue is the use of a trained classifier (e.g. LSA), which can be trained offline for many users at the same time. Given any new query, the classifier will assign an interest "id" without looking at previous URLs. The disadvantage of this approach is that it may take lots of data to train, and may incorrectly classify queries for interests it has never seen before. This is the first of various tradeoffs we need to make for anonymization. That is, we are willing to give up some memory to create more anonymous traces.

7. CONCLUSIONS & FUTURE WORK

In this paper we have illustrated some of the identifying features of search datasets. We have described two mechanisms for anonymizing users. The first is a scheme for encrypting unique queries with a hash that can be decrypted given sufficient examples of the query being used by multiple users. The benefit of the approach is that it allows the provider of the search logs to generate the hashed values in real time and reduces concerns about the privacy issues of any specific piece of the log file.

There are a number of possible future directions to explore. For example, it may be worth considering a mechanism for automatically choosing the algorithmic parameters (e.g. n and t) or even predicting the eventual number of instances of a given query. The latter, for example, may be potentially found by considering the number of results returned for the query by the search engines. It is likely that the number of results is a proxy for how identifying a certain query is (i.e. the query for "Springfield, MA" will return more hits than "Springfield, TN" in much the same way that the population counts of the first are much higher than the second). Although promising, we leave this analysis to future work.

The second technique is one for splitting users into multiple instances. By splitting based on "interests," users become dissimilar to themselves. This technique reduces the possibility of reconstructing the full user trace and finding subsets of the data that can be used to identify the user. There are also potentially more sophisticated mechanisms for generating these instances (other clustering techniques, e.g. k-means, and classification schemes, e.g. Latent Semantic Analysis), we leave their evaluation to future work. These may be able to infer the relatedness of queries in a more general way.

In the future we would also hope to apply other techniques to recover certain split features. For example, in an offline way, we can compute things like "most people who like football also like baseball." Learning such a fact would also allow us to reconnect two previously disconnected sessions.

We have discussed a number of tradeoff issues and difficulties with the anonymization schemes we propose. We continue to work on this topic and hope to develop additional anonymization mechanisms that can be applied in different ways depending on the requirements of users, corporations, and researchers. We are also considering other evaluation techniques that may more accurately determine how much information is being leaked.

We hope to provide our code for other researchers who wish to test the impacts of the various anonymization techniques described on their log analysis algorithms. It is our belief that a classification of the techniques that we have described (as well as others) in the context of real applications would provide a concrete metric for the impact of these methods on query log analysis.

8. ACKNOWLEDGEMENTS

The author would like to thank Yoshi Kohno for many useful discussions and feedback (and for running the class that spawned this project). Thanks also to Tanya Bragin for her comments and Andrew Tomkins and Ravi Kumar for access to a draft of their paper. Eytan Adar is funded by an ARCS Fellowship and a NSF Graduate Fellowship.

9. REFERENCES

[1] Arrington, Michael, "AOL Proudly Releases Massive Amounts of Private Data," last retrieved on Feb. 9, 2007 from www.techcrunch.com/2006/08/06/aol-proudly-releases-massive-amounts-of-user-search-data/

[2] "AOL Members Sue AOL LLC for Privacy Violations," Press Release, last retrieved on Feb. 9, 2007 from biz.yahoo.com/iw/060925/0166422.html

[3] AOL Research website, research.aol.com, no longer online

[4] "AOL's Massive Data Leak," last retrieved on Feb. 9, 2007 from www.eff.org/Privacy/AOL/

[5] Barbaro, Michael and Tom Zeller Jr., "A Face Is Exposed for AOL Searcher NO. 4417749," *New York Times*, August 9, 2006.

[6] Baeza-Yates, R., and B. Ribeiro-Neto, *Modern Information Retrieval*, Addison-Wesley, 1999.

[7] Bishop, M., B. Bhumiratana, R. Crawford and K. Levin, "How to Sanitize Data," WET ICE'04, Modena, Italy, June 14-16, 2004.

[8] Denning, Dorthy E., "Secure statistical databases with random sample queries," *ACM Transactions on Database Systems*, 5(3), pp. 60-80, 1980.

[9] Flegel, U., "Pseudonymizing Unix Log Files," InfraSec 2002, Bristol, UK, Oct. 1-3. 2002.

[10] Kumar, R., J. Novak, B. Pang, and A. Tomkins, "On Anonymizing Query Logs via Token-based Hashing," to appear in WWW'07.

[11] Levenshtein, V. I., Binary codes capable of correcting deletions, insertions, and reversals, Doklady Akademii Nauk SSSR, 163(4):845-848, 1965

[12] Liu, Hugo and R. Mihalcea, "Of Men, Women, and Computers: Data-Driven Gender Modeling for Improved User Interfaces," ICWSM'07, Boulder, CO, March 26-28, 2007.

[13] Menezes, A. J., P C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[14] Mogul, J. C. and M. Arlitt, "SC2D: An Alternative to Trace Anonymization," SIGCOMM 2006 Workshop on Mining Network Data, Pisa, Italy, Sep. 15, 2006.

[15] Pass, G., A. Chowdhury, C. Torgeson, "A Picture of Search" The First International Conference on Scalable Information Systems, Hong Kong, June, 2006.

[16] Padmanabhan, B. and Y. Yang, "Clickprints on the Web: Are there signatures in Web browsing data?" knowledge.wharton.upenn.edu/papers/1323.pdf

[17] "Privacy-Preserving Data mining," www.adastral.ucl.ac.uk/~helger/crypto/link/data_mining/

[18] Shamir, Adi, "How to share a secret," *Communications of the ACM*, 22(1), pp. 612-613, 1978.

[19] Shoenmakers, B., "A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting," *Lecture Notes in Computer Science*, Vol. 1666, Springer, pp. 148-164, 1999.

[20] Slagell, A., K. Lakkaraju, and K. Luo, "FLAIM: A Multi-level Anonymization Framework for Computer and Network Logs," LISA'06, Washington DC, Dec 3-8, 2006.

[21] Slagell, A., and W. Yurcik, "Sharing Computer Network Logs for Security and Privacy: A Motivation for New Methodologies of Anonymization," IEEE/CREATENET SecureComm, 2005.

[22] Sweeny, Latanya, "k-anonymity: a model for protecting privacy," *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5), pp. 557-570, 2002.

[23] Teevan, J., E. Adar, R. Jones, and M. Potts, "History Repeats Itself: Log Analysis of Repeat Queries," submitted for publication.

[24] Waters, B. R., D. Balfanz, G. Durfee, and D. K. Smetters, "Building an Encrypted and Searchable Audit Log," NDSS'04, San Diego, CA, Feb. 5-6, 2004.