# Bloom Cookies: Web Search Personalization without User Tracking

Nitesh Mor* Oriana Riva† Suman Nath† John Kubiatowicz*

*University of California, Berkeley
{mor@eecs,kubitron@cs}.berkeley.edu

†Microsoft Research, Redmond
{oriana.riva,suman.nath}@microsoft.com

***Abstract*—We propose *Bloom cookies* that encode a user's profile in a *compact* and *privacy-preserving* way, without preventing online services from using it for personalization purposes. The Bloom cookies design is inspired by our analysis of a large set of web search logs that shows drawbacks of two profile obfuscation techniques, namely profile generalization and noise injection, today used by many privacy-preserving personalization systems. We find that profile generalization significantly hurts personalization and fails to protect users from a server linking user sessions over time. Noise injection can address these problems, but only at the cost of a high communication overhead and a noise dictionary generated by a trusted third party. In contrast, Bloom cookies leverage Bloom filters as a privacy-preserving data structure to provide a more convenient privacy, personalization, and network efficiency tradeoff: they provide similar (or better) personalization and privacy than noise injection (and profile generalization), but with an order of magnitude lower communication cost and no noise dictionary. We discuss how Bloom cookies can be used for personalized web search, present an algorithm to automatically configure the noise in Bloom cookies given a user's privacy and personalization goals, and evaluate their performance compared to the state-of-the-art.**

## I. Introduction

Online services such as web search and advertising are becoming increasingly personalized. The more and the longer a service knows about an individual, the better personalization it can provide. Typically, these online services build user profiles (containing e.g., web sites frequently visited, user interests, demographics information) on the server side by tracking multiple online activities from the same user and linking them together using various techniques, usually under poorly informed user consent. In the face of privacy-concerned users and stricter privacy regulations, a search engine that provides personalized results while maintaining privacy has a definite competitive edge over other search engines. In this paper, we study how to achieve personalization while minimizing the risk of being successfully tracked by an online service, and we propose a solution called *Bloom cookies* for encoding a user's profile in an efficient and privacy-preserving manner.

The simplest way to link a user's online activities is to use the IP address of his device. However, as a device's IP address can change over time, online services track users across their IP sessions using cookies, device fingerprinting [32], and browser plug-ins (e.g., Google toolbar), to name a few. To limit such tracking, users can hide IP addresses by using techniques such as proxies and anonymity networks [33], onion routing [22], or TOR [16]. They can also disable web cookies, and browse in private mode [2] to prevent tracking by cookies. However, a fundamental problem with all these approaches is that they deny personalization because services do not have access to the information necessary for building user profiles anymore.

Although privacy and personalization are at odds, they are not mutually exclusive. For example, it is possible to maintain user profiles at the client and carry out personalization there, to the extent possible (e.g., [20], [23], [27], [47]); in this way, little or nothing is disclosed to the server. However, a pure client-side approach has serious drawbacks that make it infeasible in a real system. First, without any information about the user, the server needs to send *all* or a large number of results to the client for local personalization. The communication overhead can be prohibitive for many platforms such as mobile devices. Second, and most importantly, it requires the service to put its proprietary personalization algorithms on the client, which is often unacceptable.

To address these challenges, existing systems such as Privad [23] use two techniques. First, personalization is done by the server or by a personalization proxy and not on the client. The personalization proxy is, in general, not trusted by the client. Second, because the client does not trust the party providing personalization, it sends limited information about the user profile (e.g., high-level interests) with its request, so that the proxy (or server) can filter out results irrelevant to the user or can partially personalize the results. Hence, a key requirement of these systems is to properly obfuscate the user profiles before sending them out.

In this paper, we investigate practical techniques to obfuscate a user's profile in a way that preserves user privacy and yet allows the server (or a personalization proxy) to personalize results in a useful manner. We start with two well-known techniques for profile obfuscation: *generalization* [43] that shares items in a user's profile only at a coarse granularity (e.g., category of frequently visited web sites, instead of actual URLs), and *noise addition* [5] which adds fake items to the profile to hide the real items.

A key contribution of this paper is to systematically investigate privacy-personalization tradeoffs of such profile obfuscation techniques in the context of web search. We


NDSS '15, 8-11 February 2015, San Diego, CA, USA
 ISBN 1-891562-38-X
http://dx.doi.org/10.14722/ndss.2015.23108

use search logs from a popular search engine to quantify the tradeoffs. We find that noise addition provides a better privacy-personalization tradeoff than generalization. This is in contrast to existing systems such as Privad, Adnostic [47] and RePriv [20] that advocate for using generalized profiles to protect users' privacy. Interestingly, even though generalized profiles provide anonymity, this does not naturally translate into *unlinkability* over time. If a server is able to identify whether two requests are coming from the same or different clients (linkability), it can collect enough information to identify the user over time. On a random subset of 1300 users in our search log, even when only a user's high-level interests are disclosed, it is possible to link a user's searches across time in 44% of the cases.[1]

The superior performance of noisy profiles, however, comes at two costs. Depending on how much noise is added to the profile, a noisy profile can be very large and hence can impose a large communication overhead. Our evaluation shows that to achieve reasonable privacy and personalization, we had to add up to tens of kB of noise per request. Moreover, the noise needs to be generated by using a large noise dictionary usually provided by a trusted third party.

To address these issues, our final contribution is to propose *Bloom cookies*, a noisy profile based on Bloom filters [9][2] that is significantly smaller (comparable to the size of today's web cookies) and that does not require a noise dictionary. A Bloom cookie is generated and maintained by the client device and is sent to online services every time the user makes a service request. An online service can use the cookie to deliver personalized results. Thus, Bloom cookies can replace traditional cookies (i.e., the user can disable third party cookies in his browser), with the possibility of the user controlling what profile information is included in the Bloom cookie and when the cookie is sent to which online service.

Besides explicitly injecting noisy bits into the Bloom filter, we exploit the false positives naturally occurring in it as noise to provide privacy. We also provide an algorithm that, given a user's privacy and personalization goals, can automatically configure a Bloom cookie's parameters. Note that Bloom cookies leverage Bloom filters as a privacy-preserving data structure, in contrast to almost all previous work that adopted Bloom filters for network and storage efficiency reasons [10], [39]. To the best of our knowledge, we are the first to use Bloom filters for a practical privacy mechanism and evaluate their privacy-personalization tradeoff.

Our results show that Bloom cookies provide a more convenient privacy, personalization, and network efficiency tradeoff. For example, Bloom cookies can provide comparable unlinkability to state-of-the-art noise addition techniques with a 50% improvement in personalization, or up to $12\times$ less network overhead (2 kbit of Bloom cookies compared to 25 kbit of noisy profiles generated with state-of-the-art noise addition techniques).

[1] With a larger user population unlinkability increases, but in our evaluation we show through projection that it is still significant.

[2] A Bloom filter is a space-efficient probabilistic data structure used to store a set of elements and support membership queries. When querying if an element exists in the Bloom filter, false positives are possible but false negatives are not.

The rest of the paper is organized as follows. In §II, we define our problem space, goals and threat model, and introduce three key design questions we will answer throughout the paper. §III gives background information on web search, and defines our personalization and privacy (unlinkability) metrics. The second part of the paper answers the three design questions previously stated by showing the limitations of state-of-the-art techniques (§IV) and proposing Bloom cookies as a solution (§V). We review related work in §VI, discuss the limitations of our work in §VII, and conclude in §VIII.

## II. Privacy and personalization in web search

Our first goal is to understand how various design choices affect personalization and privacy in real systems. This understanding can help in better design of privacy-preserving personalization for many applications. To be concrete, we keep our discussion limited to web search, which we chose for three main reasons. First, search engines like Google and Bing are among the most visited web sites and users are concerned about how these services implement personalization [34]. Second, most search queries are short [25], [38] and ambiguous [15], [29], [40], and personalization can help disambiguating the queries towards an individual user's interests. Third, we had logs from a popular search engine available, making a practical analysis possible.

Like previous privacy-preserving personalization systems [20], we assume a generic client-server model. Each client is associated with a *profile* that captures the user's general preferences and is represented as a *bag* of profile items such as interest categories or URLs of web sites he frequently visits. Profiles are usually constructed using users' search history, but they could also leverage demographic information, web browsing history or social network interactions, for even richer user models. In processing a query from the client, the server utilizes the user's profile to personalize search results for him.

### A. Personalization and privacy goals

**Personalization.** Personalization in web search refers to ranking search results such that higher-ranked results are more likely to be clicked by the user than lower-ranked results. The server can use existing techniques, such as tailoring search queries to a user's interests or re-ranking search results based on web sites the user visits most frequently, in order to push likely-to-be-clicked results towards the top of the result list.

**Privacy.** We assume that the client does not trust the server with his profile. Exposing the exact profile to the server may leak a user's identity and hence for privacy, the client obfuscates his profile before sending it to the server. We consider *unlinkability* as our key privacy measure. The precise definition of unlinkability will be given in the next section; intuitively, it ensures that the server cannot identify if two queries are coming from the same client or different clients. Like previous work [5], we consider achieving unlinkability of a client's profile by obfuscating it with *noise*, i.e., by adding fake items in the profile, before sending it to the server.

### B. Threat model

We aim for unlinkability across *IP-sessions*, where an IP-session is a sequence of all queries with the same source IP address. We do not assume techniques for hiding a device's IP address (proxies and anonymity networks [33], onion routing [22], or TOR [16]) are available, because they require changes to the network infrastructure thus are not always practical. These techniques are orthogonal to Bloom cookies and can further increase a user's privacy. In our scenario, the search engine sees the IP address the search queries are coming from. Thus, our goal is to thwart a malicious server's attempts to correlate queries from different IP-sessions to find if they are associated with the same user.

Unlinkability across IP-sessions is a useful privacy goal since IP-sessions are typically short (of the order of a few weeks) in practice. For instance, a smartphone's IP address changes relatively often, basically each time there is no network activity and the radio is turned off [4]. In a home network, IP addresses change less frequently, depending on the type of provider and network contract. Bhagwan et al. [7] probed 1,468 unique peer-to-peer file sharing hosts over a period of 7 days and found that more than 50% used more than one IP address. Casado and Freedman [11] report seeing 30% of 537,790 email clients using more than one IP address in a 2-week period. According to [14], in June 2008 US machines used 5.7 distinct IP addresses in a month. Finally, another study [48] reports dynamic IPs with a volatility of 1 to 7 days, with 30% of the IP addresses changing between 1 to 3 days. In a corporate network, IP addresses may remain the same for longer, but network traffic from multiple devices is aggregated under the same IP address thus making user identification hard.[3] In general, the shorter the IP-session, the harder for the server to link different sessions. In our analysis, we assume 2-week long IP-sessions to emulate an average home network scenario based on existing studies [7], [11], [14], [48], and given our 2-month long search logs.

We assume that users' web browsers are configured in a way that prevents online services from tracking them through cookies, browser fingerprinting, browser plug-ins, or similar techniques. The browser (or the underlying system) keeps track of the user's online activities (e.g., search queries, visited sites) and maintains a profile that reflects the user's interests. The profile is not directly shared with any online service; instead it is encoded as a Bloom cookie and is sent with each search query to the server. As we later show, Bloom cookies are efficient and privacy-preserving, and yet allow the server to personalize results.

A server might launch correlation attacks based on the content of the search queries, or other meta-information associated with the queries (e.g., time of the search query, frequency, location or language). We indirectly factor the effect of such correlations in the size of our user population. A search engine potentially has billions of users, but a malicious search engine which is trying to link the different IP-sessions belonging to a single user together, can use this extra information to group search sessions into smaller clusters. A simple example is to use IP geolocation to put all the IP-sessions from a small town into one cluster. The smaller the clusters, the easier it is to link users together. In our evaluation, we use a set of 1000 users, which we believe is large enough to smoothen any of the outlier users and small enough to have a realistic and compelling use case.

Finally, we assume the server has access only to information collected though its own service (i.e., search requests submitted to the search engine). We assume the server is not colluding with other sources, such as other services (e.g., email, social networks) or third party trackers.

[3]The IP address stays the same but source ports change with every new outgoing connection. This is similar to the smartphone case where devices get a new IP address every time the radio wakes up.

### C. Key design questions

Designing a privacy-preserving personalized search engine involves many important design choices. We now discuss some important questions these choices pose. Later in the paper we answer these questions by analyzing real search logs (§IV) and show how the findings can be utilized to enable practical, privacy-preserving, and personalized web search (§V).

**Profile obfuscation mechanisms.** An important design decision is how a client's profile is obfuscated so that the server can still find it useful for personalization, but cannot link profiles from the same user. Existing solutions for privacy-preserving web search can be classified into two categories:

- *Profile generalization* [43], [49]: Profile items are generalized to a coarser granularity (e.g., a URL is generalized to its category). The server cannot distinguish between users with the same generalized profile, even if their original profiles are different. The idea has been used in other applications as well, such as cloaking a user's location with a cloaked region to achieve location privacy [3], [26].
- *Noise addition* [5], [28]: Fake profile items, called dummies, are added to and some original profile items are taken away from the profile. With a large number of fake items independently added to the profile each time it is sent to the server, two noisy profiles from the same client look different, making it difficult for the server to link them.

An important design question is:

- **What obfuscation technique is more suitable for privacy-preserving personalization of web search?**

Existing systems use these different techniques for evaluating either personalization or privacy. For example, RePriv [20], a privacy-focused system, uses generalized profiles and assumes that they can be safely shared with servers to ensure some form of anonymity. Personalization-focused systems [18], on the other hand, show that URLs without any generalization yield a better personalization. We systematically evaluate these techniques to understand their tradeoffs between privacy *and* personalization.

*Our Results.* We show that noise addition provides a better privacy-personalization tradeoff than generalization. We show that anonymity provided by generalized profiles does not naturally translate into unlinkability over time. In general, we

show that a noisy profile can provide a similar level of unlinkability as a generalized profile, but with better personalization (or similar personalization with better unlinkability). This is counter-intuitive since noise, by definition, negatively affects personalization. However, the negative effect is offset by finer granularity of profile items (than generalized profile items), resulting in a net positive improvement in personalization.

**The cost of noise.** Even though a noisy profile has its advantages over a generalized profile, they do not come for free. There are two key disadvantages. First, if many fake items must be added to the profile to ensure reasonable unlinkability, the noisy profile can be very large. Since the noisy profile is sent to the server often, possibly with each request, the communication overhead can be too much for energy-constrained devices like smartphones. Second, the fake items need to be picked from an unbiased sample of the items in the profiles of all users in the system. If the sample from which the client chooses fake items is biased (e.g., all items are related to football) and if the bias is known to the server, it can easily filter the noise out to identify the real items. Thus, the client needs to find a trusted third party who would compute an unbiased sample for him. This is a strong dependence. The sample also needs to be updated as users join and leave the system, as new profile items appear or as items' popularity changes.

This leads us to investigate the following:

- **How big a dictionary and how much noise are required to achieve reasonable unlinkability?**

*Our results.* We show that both types of cost due to noise addition are non-negligible. More specifically, the size of the noisy profile that needs to accompany each client request can be in the order of tens of kB, much larger than actual requests and responses. The overhead is significant even if the noisy profile is compressed (see §V-B).

**Efficient noisy profile.** The high costs of noisy profiles can make them impractical. Moreover, the requirement of a noise dictionary constitutes an additional threat because a malicious server may supply biased dictionaries that make the noise more predictable. The costs and additional threats of dictionaries lead us to the final question that we investigate in this paper:

- **Is it possible to receive the advantages of noisy profiles without incurring the aforementioned costs (i.e., noise dictionary and large communication overhead)?**

*Our results.* As a key contribution of the paper, we propose Bloom cookies that affirmatively answer the above question to enable a practical noise addition technique for web search. In particular, we show that Bloom cookies can achieve comparable personalization and unlinkability to a noisy profile, without requiring a noise dictionary and with an order of magnitude smaller communication overhead. We describe our solution in §V.

Note that the research questions above are in no way exhaustive, but they are some of the key questions we faced while building our system. In §IV, we answer these questions with an experimental methodology that we describe in the next section.

## III. Evaluation methodology

Our evaluation is based on search logs of a popular search engine from May and June 2013. Each entry in the search logs contains five fields: a unique user ID[4], the search query submitted by the user, timestamp, the top-10 search results shown to the user, and the results that were clicked by the user including the timestamp of each click. Each search result consists of a URL and top-3 (first or second level) ODP [1][5] categories for the web page at the URL. We replay these logs to simulate a scenario where users query a search engine, share their profile with the search engine to receive personalized results, and their IP addresses change once every two weeks (i.e., IP-session length is two weeks).

### A. Personalization strategies and metric

The state-of-the-art in web search personalization uses two main techniques for building user profiles from search logs: fine-grained **URL-based** [18] and coarse-grained **interest-based** [12], [21], [30], [37], [44] profiling. As their names suggest, URL-based profiles include URLs that users visit most often, while interest-based profiles include models of users' interests mined from users' past behavior. We implemented both techniques. To build URL-based profiles, for each search session in the user's search log where at least one of the search results was clicked, we extract the *satisfied click* [6], a click followed by a period of inactivity. We then extract the corresponding clicked URLs and assemble the user profile as a list of domain names (and not the full URLs), ordered by recurrence in the search log. To build interest-based profiles, we first label each query in the user's search log with a category. The category of a query is determined as the most common ODP category of top-10 search results of the query. Higher weights (e.g., by default double weight) are assigned to the ODP categories of the clicked results for a certain query. The interest profile of the user is then constructed as a distribution of ODP categories across all queries in the available search history for the user.

Once profiles are built, they are used for ranking search results. Specifically, for a given search query, we assign a $score$ to each of the top $M$ search results ($M = 50$ in our tests) returned for the query (note that these results are provided by the search back-end before personalization is applied, more on this later). If the domain (or any of the ODP categories) of the search result is present in the user's URL (or interest) profile, the search result receives a score of $\alpha * M$, where $\alpha$ is a parameter ranging from 0 to 1 that controls the aggressiveness of personalization. The larger $\alpha$, the more aggressive the re-ranking (we use $\alpha = 0.25$). If the domain (or the ODP category) is not present, the $score$ is 0. We then re-rank the results based on the $score$.

To evaluate personalization, we leverage user clicks recorded in the search logs. The key insight of this methodology (proposed in [18] and later widely adopted, e.g., [41])

[4]These IDs are typically established using IP address, cookies and search toolbars.

[5]The Open Directory Project (ODP) classifies a portion of the web according to a hierarchical taxonomy with several thousand topics, with specificity increasing towards the leaf nodes of the corresponding tree. Web pages are classified using the most general two levels of the taxonomy, which account for 220 topics.

is that if a personalization algorithm is able to rank "relevant" results (i.e., those that were clicked) at the top, the user will be more satisfied with the search. Hence, clicking decisions are used as a relevance metric to quantify the personalization improvements.

As in other such studies [18], [42], we measure the quality of personalization by **average rank**, defined as

$$Avg\_rank_i = \frac{1}{|R_i^c|} \sum_{r \in R_i^c} rank_r \quad (1)$$

where $R_i^c$ is the set of results clicked for a given query $i$, and $rank_r$ is the rank of the result $r$ assigned by the personalization algorithm. The smaller the average rank, the higher the personalization quality.

In evaluating personalization, the optimal case is the personalization quality provided by today's search engines to users who decide to sign in, and allow the search engine to collect their search history over the long term. This case is provided by our search logs. However, to test personalization, we also need a set of non-personalized results to be re-ranked by our personalization algorithms. We download from a separate source of the same production system where personalization is turned off (i.e., no user history is provided), the top-50 results and associated top-3 ODP categories for all queries contained in our search logs.[6] Then, for each query we compute two types of average rank: *i)* the average rank of the ideal case, $avg\_rank_{ideal}$, which is extracted directly from the search logs; and *ii)* the average rank of the personalization algorithm $test$ under study, $avg\_rank_{test}$. We then compute the absolute difference between $avg\_rank_{ideal}$ and $avg\_rank_{test}$ (i.e., if the difference is negative, it means the production system's $avg\_rank$ is smaller, which means better personalization).

Note that in our first comparison of URL-based and interest-based personalization presented in §IV-A we report the absolute drop in personalization quality compared to $avg\_rank_{ideal}$; later on we re-define our baseline as the $avg\_rank_{URL}$, our implementation of URL-based personalization, and report the percentage decrease compared to that.

### B. Privacy strategies and metrics

As described in §II-C, interest-based profiling is a form of profile generalization for privacy preservation. To represent the state-of-the-art of noise addition techniques, we implemented two techniques: RAND and HYBRID. Both these techniques work by introducing fake profile items (i.e., URLs) in the real user profile. The noise level is controlled by the parameter $f$, which represents the number of fake profile items added for each real profile item.[7] Such algorithms assume a dictionary $D$ which contains URLs and top-3 ODP categories associated to each URL. RAND represents a naïve noise addition technique, which simply draws fake URLs randomly from $D$. HYBRID is a more advanced technique inspired by [31], which draws fake URLs randomly from a user-specific dictionary, called $uD$, computed by eliminating from $D$ all URLs that do not have any ODP category matching the user's interests (which are also expressed as ODP categories). The advantage of HYBRID over RAND is that if a malicious server is able to infer a user's interests (e.g., from search keywords), it cannot simply discard (fake) URLs that do not match the user's interests.

As mentioned before, we use **unlinkability** as our privacy measure. We use two metrics of unlinkability.

*1) **Entropy-based unlinkability**:* We start from the formal definition of unlinkability given in [19], that measures the degree of unlinkability of a set of elements as entropy. A partition of the set of elements (meaning a division of the set as a union of non-overlapping and non-empty subsets) represents a possible way to "link" all elements in the set to each other (e.g., given a set of 4 elements, 15 partitions exist). In our context, "linking" means identifying user profiles collected in different contexts (e.g., different time periods) that belong to the same user. The unlinkability of the elements in the set is measured as entropy[8]

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$

where $X$ denotes the set of possible partitions and $p(x)$ is the probability mass function, $0 \leq p(x) \leq 1$, denoting the probability that $x$ is the correct partition.

Without any additional information, *a priori*, all partitions are equally possible so the probability distribution is uniform and the entropy of the elements is at its maximum ($H_{priori}(X) = -log_2(1/m)$). However, an adversary with access to some information about the partitions can, *a posteriori*, rule out some candidate partitions, thus lowering the entropy. In our context, a malicious server can observe the content of the user profiles and assign higher probabilities to certain partitions. According to [19], the *degree* of unlinkability of the set of elements against an adversary is therefore defined as the ratio between the a posteriori entropy to the a priori entropy:

$$U(X) = \frac{H_{posteriori}(X)}{H_{priori}(X)}$$

Unfortunately, this definition does not scale to a large set, as enumerating all possible partitions is a computationally hard problem. Therefore, we make some simplifying assumptions. First, we assume that we have a constant number of users in the system over time, and a user whose profile is seen in the time period $i$ (where the time period is a fixed length of time of the order of a few weeks) will have a profile also in the time period $i + 1$. Second, we assume that historical information about some users that interacted with the system is available (this allows for training of a linkability model that a potential adversary may build, see below). Third, instead of computing all possible partitions to calculate the system unlinkability, we compute "per-user unlinkability" by comparing a user's profile in time-period $i$ with all the other profiles in time-period $i+1$,

[6] As our search logs are for May-June, to ensure coverage of the results, we downloaded the data in the month of July. Queries whose clicked results were not included in the top-50 non-personalized results were eliminated from the test set.

[7] For example, if the original profile has $k$ items, the noisy profile with $f = 10$ will have $11 * k$ items.

[8] Information entropy is a well-known metric that measures the level of uncertainty associated with a random process. It quantifies the information contained in a message, usually in bits/symbol. In this setting, entropy measures the information contained in the probability distribution assigned to the set of possible partitions of the set of elements.

independently of the other users in the system, as described in details as follows.

The process consists of two steps. In the first step, we build a *linkability model* from the search logs of $n$ users over a period $T = T1+T2$ ($T = 1$ month).[9] For each of the $n$ users we create two profiles, one from the first time period $T1$ and one from the next time period $T2$. Next, to measure profile similarity we calculate the Jaccard similarity[10] between the $n^2$ possible pairs of profiles, where the first profile comes from the set of $T1$ profiles and the second comes from the set of $T2$ profiles. Using the ground truth available in the users' logs (i.e., the information of which $T1$ and $T2$ profile belong to the same user), we train a linkability model, defined as a function that maps the Jaccard similarity of a pair of profiles into the probability of these two profiles belonging to the same user (see Appendix VIII for an example of linkability function).

In the second step, we compute the unlinkability of a user's profile by calculating the a priori and a posteriori entropy. Given a set of $m$ users, where each user has two profiles computed over two consecutive (possibly overlapping) time periods $P1$ and $P2$, we apply the linkability model to compute the probability of a particular profile from $P1$ being linked to a profile in $P2$, (i.e., belonging to the same user). Note that $P1$ and $P2$ are different time periods from $T1$ and $T2$ above, but of the same length.[11] Without any information about any user, the probability of a particular profile $p_i^{P1}$ being linked to another profile $p_j^{P2}$ is $1/m$, hence, the a priori entropy is $log_2(m)$. If more information about users becomes available (by calculating the similarity between profiles and using the linkability model described above), then the probability that $p_i^{P1}$ is linked to a particular $p_j^{P2}$ changes, and we can use it to compute the a posteriori entropy, smaller than the a priori entropy. The ratio of the a posteriori to the a priori entropy is the unlinkability of user $i$.

*2) **Linkable users and max probability**:* The unlinkability metric gives an average estimation based on entropy, but it does not capture the full distribution of the a posteriori probability. Entropy-based unlinkability tries to quantify the amount of information that is required to totally break the anonymity of a profile (i.e., identify another profile with the same owner), but in practice successful attacks occur if a subset of the profiles can be linked with a good probability, significantly greater than in the case of the uniform distribution.[12] Others [13], [46] have reported similar problems with entropy-based metrics and have proposed complementing them with additional metrics such as quantiles and maximum probability.

To address this problem, we use two additional measures: linkable users percentage and max probability. *Linkable users percentage* measures the percentage of users which can be correctly linked using our linkability model. We compute the linkability probabilities between the $P1$ and $P2$ profiles of the $m$ users to obtain a $m * m$ matrix of probabilities. Using this matrix, we link each profile from $P2$ to a profile from $P1$, starting with the one with highest probability and eliminating profiles from $P1$ and $P2$ as they get linked. We define linkable users percentage as the percentage of users whose profiles of two consecutive periods can be linked correctly. *Max probability* is the maximum linkability probability in the $m * m$ matrix of probabilities after removing the top outliers, typically the top 1% (this is equivalent to computing the 99th percentile as suggested in [13]).

### C. Dataset

**Queries.** Research on search personalization has shown that personalization cannot be applied to all types of search queries successfully; personalization can improve some queries but can instead harm others [45]. For example, personalization has very little effects on navigational queries like "google" or "facebook". Instead, personalization can help ambiguous queries (e.g., one-word queries, acronyms) [41] or expanded queries [18]. To distinguish these cases, we separately report results for the entire set of queries (*all*), one-word queries (*one-word*), and expanded queries (*expanded*). Expanded queries are queries that at the beginning of a search session contained only one or two words and by the end of the search session were expanded into several words. As an example, the query "ndss" was expanded into "ndss security" which was expanded into "ndss security conference 2015". If a click was reported for a result shown in the third query's result page, we are interested in evaluating whether when the first query is submitted, personalization can rank the clicked result higher than it appeared in the first query's result page.

**Users.** In measuring personalization, we selected users that had a search history long-enough to build reasonable user profiles. We selected users from the month of June 2013 that had at least 250 queries in that month and whose resulting profile had at least 22 URL domains and 11 interests. For users with more than 22 URL domains and 11 interests we used the top 22 and top 11, respectively, so the profile length was the same for all users. We selected 308 users, for a total of 264,615 search queries. User profiles were built using 2 consecutive weeks of search history, while the following third week was used for testing. Using a sliding window, we also tested the queries in the fourth week. In evaluating privacy, we used a larger dataset consisting of 1300 users (300 users from May 2013 and 1000 from June 2013) for a total of 331,361 search queries. These users included the 308 users selected for personalization. For evaluating privacy a larger set of users could be used because no constraints needed to be imposed on the length of users' search history.

[9] In the actual experiments, to train the linkability model we used $n = 300$ users from the May 2013 logs ($T1$ is May 1–15 and $T2$ is May 16–30) with a total of 66,746 queries.

[10] The Jaccard similarity coefficient (or Jaccard index) measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets. In our case the sample sets are the user profiles. Each user profile is in fact a set of URLs or interests.

[11] We used $m = 1000$ users from the June 2013 logs, with a total of 264,615 queries. $P1$ is June 1–14 and $P2$ is June 8–21.

[12] To illustrate, let us consider 4 profiles a user can be linked against. The a priori probability is 0.25 for each profile. Now, let us assume that the a posteriori probabilities are either a) [0.05, 0.45, 0.45, 0.05] or b) [0.115, 0.115, 0.655, 0.115]. The entropy for a) and b) is similar (1.469 and 1.476 respectively), however it is easier to link one of the profiles in case b) (assuming the 3rd profile is the correct one). Although, the average unlinkability is the same, the number of correctly identified users is possibly larger for b.

## IV. RESULTS

We now answer the design questions we posed in §II-C.

### A. Limitations of generalized profiles

We first report how generalized profiles perform under our evaluation. For simplicity, we first compare them with "exact" profiles without any noise, i.e., profiles consisting of the URLs frequently visited by users. This analysis will give us a lower bound of unlinkability and upper bound of personalization of noisy profiles as noise can only increase unlinkability and hurt personalization of the exact profile. Later we evaluate noisy profiles as well.

Table I compares personalization and privacy of exact and generalized profiles. For personalization, we report the difference between the $avg\_rank$ of production-quality personalization (called $avg\_rank_{ideal}$ in §III) and the one obtained when ranking results using our URL or interest-based personalization algorithms. For privacy, we compute entropy-based unlinkability, linkable users percentage and max probability.

All personalization values in Table I, including exact profiles, are negative, which means that our personalization algorithms perform worse than the production-quality algorithm (i.e., $avg\_rank_{ideal}$ is smaller than the $avg\_rank$ obtained with our personalization). This is expected as our algorithms for user profiling and personalization are not optimized and certainly not advanced as those used in the commercial search engine. Moreover, they most likely use a shorter user history.

However, this negative performance does not affect our evaluation because we are interested in evaluating the *relative loss* in personalization when privacy protection is enabled.

We make two observations from the results in Table I. First, generalized profiles significantly hurt personalization. The average rank with generalized profiles is from 24% (-2.14 vs -1.73 for "all") to 82% (-1.78 vs -0.98 for "expanded") worse than that with exact profiles, mainly because generalized profiles contain less information for personalization. Other studies on personalized search (e.g., [18]) drew a similar conclusion and emphasized the need for exact URLs in the profiles.

Second, as expected, generalized profiles provide better unlinkability than (noise-free) exact profiles, but they still do not ensure reasonable unlinkability. In other words, even though anonymity of generalized profiles make linking consecutive IP-sessions of the same user harder, user tracking is still largely achievable—in about 44% of the cases.

Because of the limitations above, we argue that generalized profiles are not suitable for privacy-preserving personalization of web search. Exact profiles do not ensure unlinkability either, but they are promising because they allow us to add noise. Next we show that it is possible to add noise to increase unlinkability without substantially hurting personalization.

**Why do generalized profiles perform poorly?** We took the 1000 users used in the analysis above and divided their search traces into two 2-week time periods. For each time period, we extracted their interest profile. We then computed *a)* the Jaccard similarity between the profiles of the same user from the two time periods, and *b)* the Jaccard distance between each user's profile (from the first time period) from the average profile. The average profile was computed by treating the traces of all users (from the first time period) as one single user's trace and extracting the URL and interest profiles. Figure 1 reports the CDFs for *a)* and *b)*. We distinguish between users whose profiles were correctly or incorrectly linked across the two time periods (this corresponds to the "linkable users percentage" metric in Table I).

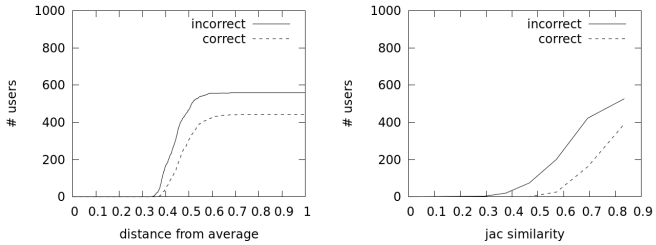



**Fig. 1: Jaccard distance of the interest-based profile of each user from an average profile computed across all users (a), and Jaccard similarity for each user's 2-week long interest-based profiles over 4 weeks (b). Users are grouped into correctly/incorrectly linked (1000 users).**

Correctly linked user profiles have on average the same distance from the average profile as incorrectly linked user profiles (graph on the left side). In fact, the curves for incorrectly and correctly linked profiles saturate at the same point (around a distance of 0.55). This shows how interests are good at hiding users with unique profiles items, thus making them less likely to be linked. Although this may seem intuitive, the distribution of interests across all users is not uniform and a large fraction of interests are unique among users. For instance, we observed that the 20 top interests across all 1000 users are common to 20–70% of the users, but then there is long tail of interests which are unique to a handful of users. At a given point in time, unique interests are not sufficient to uniquely identify users, but over time they make users linkable. In other words, anonymity helps make users unlinkable, but it is not sufficient because the similarity between a user's profiles from different time periods can make them linkable. This is shown in the graph on the right side: profiles whose similarity across the two time periods is above 0.7 are likely to be linked (between 0.7 and 0.8 the curve of linked users shows a steep increase) while profiles whose similarity is below 0.65 are likely not to be linked.

### B. Benefits and costs of noisy profiles

We now consider the effect of adding noise to exact profiles using state-of-the-art techniques represented by RAND and HYBRID described in §III-B. Table II evaluates the privacy protection (measured as entropy-based unlinkability, linkable users percentage and max probability) provided by RAND and HYBRID as well as their impact on personalization and network efficiency. The noise level of RAND and HYBRID is controlled by the parameter $f$, which represents the number of fake profile items added for each real profile item. Both the algorithms assume a dictionary $D$ which contains 157,180 URLs and top-3 ODP categories associated to each URL. Tests are executed on the same dataset and using the same methodology as for

| Type of profile | Absolute personalization loss compared to production quality | | | Unlinkability | | |
|---|---|---|---|---|---|---|
| | all | one-word | expanded | entropy-based | % linkable users | max prob (1%) |
| Exact (URLs) | -1.73 | -0.83 | -0.98 | 0.66 (0.12) | 98.7 | 0.73 |
| Generalized (Interests) | -2.14 | -1.32 | -1.78 | 0.83 (0.06) | 44.1 | 0.37 |

**TABLE I: Personalization-privacy tradeoff for exact and generalized profiles. For personalization, the table reports the difference between $avg\_rank_{ideal}$ (extracted from the search logs with production quality personalization in use) and $avg\_rank_{URL}$ and $avg\_rank_{Interest}$ (obtained using our unoptimized URL and interest-based personalization algorithms). Results are for "all" queries (308 users, 264,615 queries), with a breakdown "one-word" (44,351) and "expanded" (146,497) queries. For privacy, it reports unlinkability as avg (stdev) entropy-based unlinkability, linkable users percentage and max probability with top 1% outliers removed. Privacy results are computed for 1000 users (264,615 queries).**

| Noise addition mechanism | | Personalization loss (%) compared to exact profiles | | | Unlinkability | | | Avg profile size (bits) |
|---|---|---|---|---|---|---|---|---|
| | noise level | all | one-word | expanded | entropy-based | % linkable users | max prob (1%) | |
| Exact profile (URLs) | f=0 | 0.00 | 0.00 | 0.00 | 0.66 (0.12) | 98.7 | 0.73 | 294.0 |
| RAND | f=10 | 0.14 | 0.11 | 0.28 | 0.69 (0.09) | 97.4 | 0.63 | 3784.5 |
| | f=20 | 0.36 | 0.96 | 0.56 | 0.76 (0.07) | 93.1 | 0.47 | 7273.2 |
| | f=30 | 0.49 | 0.49 | 0.68 | 0.81 (0.06) | 80.2 | 0.37 | 10763.3 |
| | f=40 | 0.64 | 1.41 | 1.00 | 0.88 (0.05) | 57.0 | 0.30 | 14253.9 |
| | f=50 | 0.81 | 1.83 | 1.32 | 0.92 (0.02) | 34.1 | 0.15 | 17743.1 |
| | f=60 | 0.89 | 1.19 | 1.56 | 0.94 (0.02) | 26.8 | 0.15 | 21232.6 |
| | f=70 | 1.10 | 1.61 | 1.82 | 0.96 (0.01) | 20.0 | 0.06 | 24722.6 |
| HYBRID | f=3 | 0.71 | 1.49 | 1.15 | 0.68 (0.10) | 91.2 | 0.67 | 1183.3 |
| | f=5 | 1.15 | 1.99 | 1.78 | 0.75 (0.07) | 73.3 | 0.48 | 1773.9 |
| | f=7 | 1.76 | 2.95 | 2.89 | 0.79 (0.05) | 47.3 | 0.25 | 2366.2 |
| | f=10 | 2.15 | 3.53 | 3.42 | 0.81 (0.05) | 34.9 | 0.19 | 3252.3 |
| | f=15 | 3.55 | 4.73 | 5.80 | 0.83 (0.05) | 17.4 | 0.16 | 4727.5 |
| | f=20 | 4.23 | 6.52 | 7.17 | 0.83 (0.06) | 13.6 | 0.24 | 6197.9 |

**TABLE II: Personalization, privacy and efficiency tradeoffs for RAND and HYBRID when varying the noise level $f$ (1000 users, 264,615 queries). For personalization, the table reports the difference between $avg\_rank_{URL}$ (computed using exact profiles, first row in the table) and the average rank obtained with URL-based noisy profiles. For privacy, it reports unlinkability as avg (stdev) entropy-based unlinkability, linkable users percentage and max probability with top 1% outliers removed. For efficiency, it reports the size of the noisy profile.**

the results in Table I. Personalization results are reported as loss compared to the average rank obtained with exact profiles; later we also consider generalized profiles and compare results in Table I and Table II. Finally, note that for RAND and HYBRID $f$ varies on a different range of values. As we will show below, this is necessary because the algorithms work in a different manner, so that to obtain the same performance different amounts of noise are required.

Both noise addition techniques are able to provide higher unlinkability than exact profiles. Compared to exact profiles where 98.7% of user profiles were correctly linked, noise addition lowers the percentage to 20% (with RAND) or 5.8% (with HYBRID). Notice that although the average unlinkability is generally better for RAND, in practice HYBRID makes users less linkable, as shown by the linkable users percentage and the max probability metrics, which are both smaller than with RAND. The reason for this behavior is what we discussed in §III-B and this is why we do not consider the average entropy-based metric alone.[13]

[13]Even when entropy-based unlinkability is high (e.g., 0.88 for RAND with $f = 40$ vs. 0.81 for HYBRID with $f = 30$, a larger number of profiles (57% vs. 40%) may be linked correctly because of the probability distribution. This behavior is captured by the max probability metric (0.30 vs. 0.19)–the higher the max probability, the more likely profiles are correctly linked.

The two algorithms have distinct behaviors. RAND is a conservative algorithm that provides only moderate unlinkability but keeps the personalization loss relatively small. To achieve levels of unlinkability comparable to HYBRID, RAND requires much larger amounts of noise (this is why for RAND we consider up to the case of $f = 70$), thus significantly increasing the network overhead. HYBRID is a more aggressive and efficient algorithm which achieves high levels of unlinkability with smaller amounts of noise, but with a big loss in personalization. The reason behind this is that in HYBRID the added noise relates to true interests of the user (i.e., it has the same ODP categories as the true URLs) thus having a probability of collusion with the URLs that interest the user higher than with RAND.

**Comparison with generalized profiles.** Although HYBRID causes a decrease in personalization, this is still much smaller than with interest-based profiles. We can combine Table I (second row) and Table II to directly compare noisy profiles to generalized profiles. For HYBRID with f=20, for instance, the personalization loss for "all" is 4% compared to the personalization quality of exact profiles, while interest-based profiles have a decrease of 24% compared to the same exact profiles. For "expanded" queries the difference is even larger: a loss of 7% with HYBRID-20 and a loss of 82% with interest-based profiles. Summarizing, the comparison shows that noisy

profiles, RAND with $f \geq 50$ and HYBRID with $f \geq 10$ can simultaneously provide better personalization and better unlinkability than generalized profiles. For example, HYBRID with $f = 10$ links 35% users at the cost of a personalization loss of $< 4\%$, while generalized profiles link 44% people at the cost of a personalization loss of 24–82%.

**Costs of noisy profiles.** Adding noise to profiles inflates their sizes and requires a noise dictionary. Although HYBRID requires less noise, the network overhead these noise addition techniques cause is substantial. As an example, a web cookie is typically a few hundred bytes in size, and no more than 4kB. With a modest level of noise such as $f = 30$, the size of the profile is more than 30 times the noise-free profile and several times the size of a typical web cookie. HYBRID, however, requires a larger dictionary as it needs both URLs and categories of the URLs. In our evaluation, the dictionary sizes of RAND and HYBRID were a few MBs. The dictionaries require a trusted third party (as mentioned in §II-C), and their network and memory footprints are significant.

## V. Bloom Cookies

We now describe our solution for building noisy profiles that have similar unlinkability and personalization advantages to RAND and HYBRID, but without their costs. Our solution, which we call Bloom cookies, is significantly smaller in size than RAND and HYBRID and does not require any noise dictionary. In this section, we discuss the design of Bloom cookies, compare their performance with RAND and HYBRID, and present an algorithm to automatically tune their parameters for target privacy and personalization goals.

### A. Bloom cookies design

Bloom cookies are based on Bloom filters [9], a well-known probabilistic data structure. A Bloom filter is used to store elements from a set $E$, and is implemented as a bit-string of size $m$ with $k$ hash functions. When querying if an element exists in the Bloom filter, false positives are possible but false negatives are not. The probability $p$ of false positives can be controlled by varying $m$ and $k$; according to [10], $k = m/n \cdot ln2$ minimizes $p$, where $n = |E|$.

One straightforward way to use Bloom filters is to insert the URLs from the noisy profile generated by RAND or HYBRID into a Bloom filter, which the client sends to the server along with his queries. For personalization, the server simply queries the Bloom filter for all the URLs contained in the search results for the submitted search query and re-ranks the results accordingly. The number of search results to be re-ranked is commonly in the range 10–100, which makes the number of Bloom filter queries acceptable. As the Bloom filter size can be significantly smaller than the actual list of URLs, this can reduce the communication overhead. However, this approach still does not remove the need for a noise dictionary required by RAND and HYBRID.

To avoid the need for a noise dictionary and reduce even further the communication overhead, we introduce noise at the bit-level of a Bloom filter. More specifically, we start with the exact profile of the client, encode the URLs present in the exact profile into a Bloom filter, and then set a random set of *fake bits* in the filter to 1. We call this data structure, consisting of a Bloom filter of an exact profile and a set of fake bits, a *Bloom cookie*. The presence of fake bits increases the false positive rate of the filter and acts as noise. The number of fake bits acts as a tuning knob to control the magnitude of noise.

The above use of Bloom filters to generate Bloom cookies is relatively simple. However, unlike almost all previous work that adopted Bloom filters for network and storage efficiency reasons [10], [39], Bloom cookies use them as a privacy-preserving data structures. To the best of our knowledge, we are the first to use Bloom filters for a practical privacy mechanism and to evaluate its privacy-personalization tradeoff. The only other work in this direction we are aware of is [8], which is discussed in §VI.

We argue that there are at least five benefits that make Bloom filters interesting for profile obfuscation.

1) *Efficiency*: In terms of size, Bloom filters are much more compact than a bag of URLs used by noise addition techniques such as RAND and HYBRID. This reduces the communication overhead of sending noisy profiles to the server.
2) *Noisy by design*: Bloom filters' false positives, typically considered as drawbacks, are an advantage for us. In fact, the false positives in a Bloom filter act as natural noise that can be controlled via various design parameters such as the number of hash functions.
3) *Non-deterministic noise*: The level of noise introduced by Bloom filters changes automatically as the content of the filter changes. This makes it harder for an adversary to predict the level of noise utilized. As discussed in [5], noise determinism is a significant problem for standard noise addition techniques.
4) *Dictionary-free*: By adding noise by setting random fake bits, Bloom cookies can work without any noise dictionary. As discussed in §II-C, the requirement of a noise dictionary introduces additional overhead and privacy threats.
5) *Expensive dictionary attacks:* Unlike most profile obfuscation techniques that represent noisy profiles as a list of profile items, Bloom filters represent them as an array of bits. To build a complete user profile, a potential adversary would need to query the Bloom filter for all possible elements.

In addition to false positives naturally occurring in Bloom filters, we inject noise by setting random bits in the filter. The level of noise is controlled by the parameter $l$ (different from the noise level $f$ used in RAND and HYBRID) which represents the *fraction of bits* set in a Bloom filter, either corresponding to the original profile items or to noise. Note that $l$ is used to control only the number of fake bits that we need to set after we insert the original profile items. If the number of bits set by the original profile items is already greater than the target value of $l$, we do not add any noisy bits. In our tests, we did not face this situation except for $l = 0$. The reason for configuring noise as a fraction of the total number of bits (rather than as a constant number of bits) is to keep the number of bits observed by the server constant. To understand let us consider the case of two profiles $A$ and $B$ each containing $n = 10$ element. When stored in a Bloom filter (with $k = 1$), let us assume $A$ sets 7 unique bits and $B$ sets 10 unique bits.

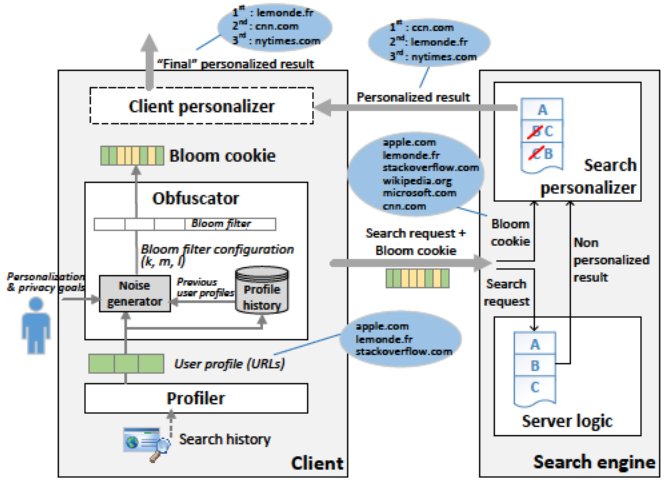



**Fig. 2: Bloom cookies framework for web search.**

It is intuitive to see that when trying to reverse engineer the content of the Bloom filter the second profile will be mapped to more profile items than the first one, thus indicating that $B$ requires fewer noisy bits than $A$ to achieve the same level of protection. This is why the fraction of bits set in the Bloom filter, regardless of whether they are real or fake bits, is itself an indication of the degree of obfuscation.

Figure 2 summarizes the Bloom cookies framework applied to web search. At the client, a profiler builds a personalization profile based on the user's search history. The profile, consisting of URLs of the web sites the user visited most often, is fed into an obfuscator which generates a Bloom cookie for the ongoing IP-session. The obfuscator configures the level of obfuscation of the profile through two parameters: number of hash functions ($k$) and noise level ($l$). Note that, in principle, a Bloom filter's probability of false positives $p$ depends on $k$ and $m$ (the larger $k$, the smaller $p$; the larger $m$, the smaller $p$ [5]). In practice, as shown below, $m$ has little impact on the obfuscation achieved, so we decide to vary only $k$ for obfuscation purposes. $m$ is typically set to 1000–2000 bits. The parameters $k$ and $l$ are computed by a noise generator once for each IP-session. This uses a simple prediction algorithm (described later) that given a user's personalization and privacy goals together with a history of profiles previously sent to the server predicts the optimal $<k, l>$ configuration. The Bloom cookie is sent to the search engine along with the user's search request, and stays constant for the current IP-session. At the search engine, a personalizer re-ranks the search results based on the noisy user profile stored in the Bloom cookie. Optionally (we did not implement this component yet), a client-side personalizer can further refine the results and their rank based on the noise-free user profile which is known to the client.

### B. Comparison with state-of-the-art obfuscation techniques

We now evaluate how effective Bloom cookies are as obfuscation mechanisms compared to state-of-the-art noise addition techniques, particularly RAND and HYBRID. For simplicity, we use a Bloom filter with $m = 2000$ and $k = 3$, and vary only the level of noise $l$. In the next set of tests, we study its performance when also varying $k$ and $m$. For RAND and HYBRID the noisy profile is in plain text. For BloomCookies we consider the worst case scenario in which a malicious server reverse engineers the content of the filter (this means querying the filter for all possible profile items present in the dictionary of RAND) in an attempt to play a linkability attack. Note that this brute-force reversal of the filter, although expensive, is strictly a stronger attack than intersection attacks or similar, which may be easier with Bloom filters. Table III reports the privacy protection, personalization loss and network overhead of BloomCookies for increasing levels of noise. Tests are executed on the same dataset and using the same methodology as for the results in Table II.

Overall, Bloom cookies provide a more convenient privacy-personalization-efficiency tradeoff. If we compare BloomCookies to HYBRID (see Table II) and consider the cases $l = 25$ and $f = 15$ respectively, we see that not only do BloomCookies provide comparable (actually slightly better) unlinkability, they also halve the personalization loss of HYBRID (1.77% vs. 3.55% for "all"). If we compare BloomCookies to RAND, to achieve the same unlinkability guarantees of BloomCookies, RAND needs to add lots of noise. In fact, only for $f = 70$ (i.e., 70 fake profile items for each real profile item) RAND achieves a level of unlinkability comparable to BloomCookies with $l = 25$ (the average unlinkability is 0.96 and 0.95, respectively), while personalization is slightly better for RAND. On the other hand, the size of the RAND profile for $f = 70$ is 24.7 kbits compared to the 2 kbits of BloomCookies, more than 20 times bigger.

As we discussed above, one may argue that a simpler solution would be storing the output of RAND (or HYBRID) in a Bloom filter. We experimented with this option by using a number of hash functions that minimizes the probability of false positives such that the unlinkability and personalization performance remain the same. For space constraints, we only comment on the case of RAND with $f = 70$ stored in a Bloom filter where we obtain a profile of size of 15.6 kbits, still almost 8 times larger than BloomCookies. Moreover, this alternative approach does not remove the dependency on the noise dictionary.

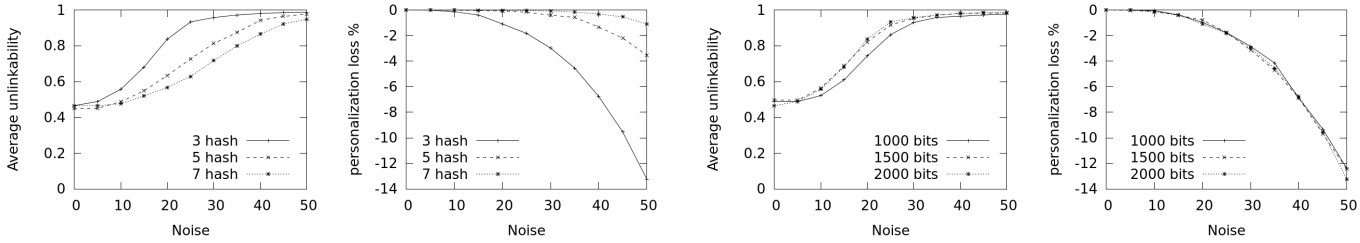
If we compare the performance of BloomCookies to exact URLs (no noise) or interest-based profiles (Table I), the noisy profile provided by BloomCookies clearly provides a much better privacy-personalization-efficiency tradeoff. Interest-based profiles drop the level of personalization achieved with exact URLs by 24% ("all") to reduce the percentage of linkable users from 99% to 44%. With the same drop in personalization as interest-based profiles, BloomCookies can reduce the linkable users to 0.3%; likewise, with a drop in personalization of only 1% (still compared to exact URL profiles), BloomCookies ($l = 15$) can provide a similar level of unlinkability (45% vs 44% linkable users) as interest-based profiles. All this is achieved with a network overhead comparable to that of today's web cookies.

### C. Effects of k and m

We now verify the assumption that $k$ (number of hash functions) and $l$ (level of noise) are the main parameters to control obfuscation in Bloom cookies. Figure 3 reports the average unlinkability and personalization loss compared to

| Noise addition mechanism | | Personalization loss (%) compared to exact profiles | | | Unlinkability | | | Avg profile size (bits) |
|---|---|---|---|---|---|---|---|---|
| | noise level | all | one-word | expanded | entropy-based | % linkable users | max prob (1%) | |
| Exact profile (URLs) | f=0 | 0.00 | 0.00 | 0.00 | 0.66 (0.12) | 98.7 | 0.73 | 294 |
| BloomCookies | l=0 | 0.16 | 0.04 | 0.35 | 0.65 (0.11) | 93.8 | 0.71 | 2000 |
| | l=5 | 0.17 | 0.08 | 0.37 | 0.66 (0.10) | 91.5 | 0.68 | 2000 |
| | l=10 | 0.25 | 0.42 | 0.51 | 0.73 (0.08) | 91.1 | 0.55 | 2000 |
| | l=15 | 0.57 | 1.02 | 1.07 | 0.80 (0.07) | 80.0 | 0.41 | 2000 |
| | l=20 | 1.00 | 1.61 | 1.83 | 0.90 (0.05) | 44.7 | 0.29 | 2000 |
| | l=25 | 1.77 | 2.42 | 3.01 | 0.95 (0.01) | 15.6 | 0.08 | 2000 |
| | l=30 | 3.30 | 6.75 | 5.70 | 0.97 (0.00) | 2.3 | 0.01 | 2000 |
| | l=35 | 4.68 | 6.50 | 7.73 | 0.98 (0.00) | 2.5 | 0.02 | 2000 |
| | l=40 | 6.81 | 11.75 | 11.62 | 0.99 (0.00) | 1.0 | 0.00 | 2000 |
| | l=50 | 11.94 | 19.68 | 20.66 | 0.99 (0.01) | 0.3 | 0.05 | 2000 |

**TABLE III: Personalization, privacy and efficiency tradeoffs for Bloom cookies when varying the noise level $l$ (1000 users, 264,615 queries). For personalization, the table reports the difference between $avg\_rank_{URL}$ (computed using exact profiles, first row in the table) and the average rank obtained with Bloom cookies. For privacy, it reports unlinkability as avg (stdev) entropy-based unlinkability, linkable users percentage and max probability with top 1% outliers removed. For efficiency, it reports the size of the noisy profile.**




(a) Effect of $k$ and $l$

(b) Effect of $m$ and $l$

**Fig. 3: Bloom cookies average unlinkability and personalization loss when varying $k$ ($k = 3, 5, 7$) or $m$ ($m = 1000, 1500, 2000$) with different noise levels ($l$), 300 users.**

exact URL profiles when varying $k$ ($k = 3, 5.7$) or the size of the Bloom filter $m$ ($m = 1000, 1500, 2000$), and for different levels of noise ($l$). When increasing $k$ (see Figure 3(a)), the average unlinkability decreases and the personalization loss shrinks. In fact, increasing $k$ means reducing the probability of false positives as well reducing the number of noisy bits controlled by $l$ (because with larger $k$ more bits are set for each profile item when stored in the Bloom filter). Conversely, increasing $m$ (slightly) increases unlinkability (and has almost no effect on personalization). This is because although a larger $m$ reduces the probability of false positives, a larger Bloom filter means that more noisy bits are set because $l$ controls the *fraction* of bits set. The noise effect controlled by $l$ prevails thus making unlinkability higher when $m$ increases. We notice however that the variations are relatively small (there is no substantial difference between the case $m = 1500$ and $m = 2000$ bits). For this reason, we use only $k$ and $l$ to control the noise level of BloomCookies.

### *D. Algorithm for configuring Bloom cookies*

By varying the noise level, a user can control the privacy-personalization tradeoff. A privacy-concerned user may choose to operate at the highest noise levels, while a user that values privacy and personalization in the same manner may decide to operate at moderate noise levels. We designed an algorithm that automatically configures the noise parameters in the Bloom cookie given a user's privacy and personalization goals.

**Algorithm.** The pseudocode of the algorithm is shown in Figure 4. The algorithm takes as input a personalization goal specified as maximum percentage loss that a user is willing to tolerate (compared to the personalization obtained with exact profiles), and a privacy goal specified as the minimum unlinkability a user wants to achieve. In addition, the algorithm uses the history of profiles previously sent by the client to the server to compute the profile similarity over time. The algorithm returns the pair $<k, l>$ for configuring the Bloom cookie.

The algorithm employs two prediction models, one for personalization and one for privacy. The models are trained using a set of users for which search history is available.[14]

We build the *personalization model* by computing the loss in personalization for the training users when independently varying the parameters $k$ and $l$ ($m = 2000$). Given a target personalization loss, the model predicts various $<k, l>$ combinations by performing a linear interpolation between all measured data points.

[14]We train the models using 300 users for which we have 1 month-long logs (300 are used for training the privacy model and a subset of 93 users whose personalization profile is large enough is used for the personalization model).

```
def find_noise(sim,pergoal,privgoal,permodel,privmodel):
  simc = find_similarity_class(sim)
  solutions = []
  for k in k_vals:
    per = scaling(interpolation(permodel[k]))
    priv = scaling(interpolation(privmodel[simc][k]))
    # find _min s.t. priv(_min) = privgoals
    _min = inverse(priv)(privgoal)
    # find _max s.t. per(_max) = pergoal
    _max = inverse(per)(pergoal)
    if _min<=_max: solutions.append((k,_min))
  return random.choice(solutions)
```

**Fig. 4: Code snippet of the algorithm for configuring a Bloom cookie's obfuscation parameters ($k$ and $l$) given a personalization and a privacy goal.**

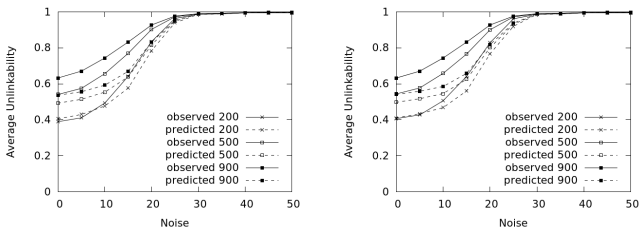



**Fig. 5: Error in predicting unlinkability when increasing the number of users from 100 to 900 (a) and from 50 to 900.**

To build the *privacy model*, we leverage the observation that the similarity of a user's profiles over time makes him more trackable. Hence, the greater the similarity, the more noise required to achieve a certain level of unlinkability. The goal of the privacy model is then to represent the relationship between similarity, unlinkability and $<k, l>$. We compute the Jaccard similarity between two consecutive 2-week profiles of the training users and then divide them in $s = 10$ buckets based on the similarity value.[15] For each bucket, we then create a privacy model by doing a linear interpolation in a similar way as for the personalization model. For a desired level of unlinkability the model predicts the pair $<k, l>$. Thus, given a privacy goal and the similarity of a user's profile across time, the algorithm finds which similarity bucket the user belongs to, and then uses the appropriate privacy model for that bucket to predict $<k, l>$.

The privacy model provides a lower bound on the noise (i.e., with more noise higher unlinkability is achieved). The personalization model provides an upper bound (i.e., with more noise a larger personalization loss is experienced). If the lower bound is higher than the upper bound, there is no solution satisfying the goals. Otherwise, the solution is determined by randomly selecting a $k$ among the possible values and use the minimum noise level for such $k$.

**Effect of population size.** Unlinkability depends on the population size. Hence, if we train on a certain sized set of users and use a different sized population for testing, we need to take this factor into account. When a user population increases we expect it to be harder to link user profiles so the unlinkability goal becomes easier to some extent.

We hypothesize that when the population size increases from $n$ to $kn$, the probability distribution function of different profiles from different time periods to belonging to the same user (i.e., what we called p(x) in §III-B) remains the same, except scaling by a constant factor. In this case, we can express the new unlinkability $u'$ for that user in terms of the old unlinkability $u$ and the population sizes. Given the unlinkability definition in §III-B, unlinkability for a population of size $n' = kn$ is the following

$$u' = \frac{\sum^{n'}(p'_i * \log p'_i)}{\log \frac{1}{n'}} = \frac{k\sum^{n}(\frac{p_i}{k} * \log \frac{p_i}{k})}{\log \frac{1}{kn}} = \frac{u * \log \frac{1}{n} - \log(k)}{\log \frac{1}{kn}}$$

We verify our hypothesis by taking user populations of different sizes ($n = 50, 100$) and use them to predict the average unlinkability for larger user populations ($kn = 200, 500, 900$). We measure the error for varying levels of noise. Results are shown in Figure 5. The average prediction error for $n = 100$ is 2.8%, 5.8% and 7.7% for scaling to 200, 500 and 900 users, respectively. For $n = 50$, it is 3.2%, 6.3% and 8.1% when scaling to 200, 500 and 900 users, respectively. The prediction error is reasonably small for small scaling factors. However, even for large scaling factors, the prediction is typically conservative—the observed unlinkability is larger than the predicted value. Hence, we use this approach to make our algorithm's predictions scale to larger populations of users.

**Algorithm evaluation.** We evaluate the performance of the algorithm on a set of 700 users (and a subset of size 215 for personalization). For a particular combination of privacy and personalization goals, we invoke the algorithm for each user to obtain a $<k, l>$ pair for that user. We then measure the average unlinkability and personalization loss for such a set of users assuming they would use the noise parameters as specified by the algorithm. We then verify whether the initial goals are met. Table IV reports the results for various combinations of goals.

Privacy goals are met in all conditions, with an average unlinkability higher than desired. Personalization goals are met fully or with a very small error when the minimum personalization loss requested is below 0.3%. For instance, for a target personalization loss below 0.2%, the actual loss is 0.29% and 0.34%. Personalization strongly depends on the user so it is harder to build a general model which works well. More information about the user profile (e.g., distribution of user interests, demographics, etc.) could be used as features for training the personalization model. On the other hand, simply given a larger training set, we expect the algorithm to be able to meet stricter goals on personalization as well.

## VI. Related Work

The body of research closest to our work is that of client-side solutions for preserving privacy in services such as search, advertising, and news portals [20], [23], [27], [47]. There are three problems with current client-side solutions: the client

[15] Instead of imposing arbitrary similarity ranges, we derive the ranges by diving the users in equal sets, so to mimic the actual user distribution. As an example, the first 3 classes are (0.0, 0.207), (0.207, 0.313) and (0.313, 0.399).

| Desired goals | | Achieved goals | |
|---|---|---|---|
| personalization | privacy | personalization | privacy |
| 0.2 | 0.7 | × (0.29) | ✓(0.84) |
| 0.2 | 0.8 | × (0.34) | ✓(0.93) |
| 0.2 | 0.9 | No solution | |
| 0.3 | 0.7 | × (0.31) | ✓(0.83) |
| 0.3 | 0.8 | × (0.34) | ✓(0.95) |
| 0.3 | 0.9 | ✓(0.29) | ✓(0.93) |
| 0.4 | 0.7 | ✓(0.29) | ✓(0.84) |
| 0.4 | 0.8 | ✓(0.34) | ✓(0.95) |
| 0.4 | 0.9 | × (0.48) | ✓(0.91) |
| 0.5 | 0.7 | ✓(0.31) | ✓(0.84) |
| 0.5 | 0.8 | ✓(0.34) | ✓(0.94) |
| 0.5 | 0.9 | ✓(0.50) | ✓(0.93) |
| 0.6 | 0.7 | ✓(0.31) | ✓(0.84) |
| 0.6 | 0.8 | ✓(0.42) | ✓(0.95) |
| 0.6 | 0.9 | ✓(0.50) | ✓(0.93) |
| 0.7 | 0.7 | ✓(0.31) | ✓(0.84) |
| 0.7 | 0.8 | ✓(0.50) | ✓(0.95) |
| 0.7 | 0.9 | ✓(0.68) | ✓(0.95) |

**TABLE IV: Algorithm performance (700 users). The table reports the desired goals and if a solution is found whether the goals were met. In parenthesis it is the actual unlinkability and personalization achieved with the algorithm's predicted $k$ and $l$.**

overhead generated by carrying out personalization at the client, the quality of personalization achievable by sharing coarse-grained profiles with servers, and the privacy loss, particularly in the case of an adversary that attempts to link a user's activities over time. Bloom cookies are designed to help with all three issues.

In our work, we do not rely on anonymization proxies, onion routing, TOR and other anonymity networks because their deployment may be too expensive. However, if these solutions are available, Bloom cookies become even more important for personalization because servers are left with no means of building user profiles.

Non-anonymous privacy models that allow user identification, but prevent derivation of personal information have been largely studied in the privacy community, in particular for web search. Tools like TrackMeNot [24], PRAW [36], PDS [31], OQF-PIR [35], GooPIR [17] and NISPP [50] (all extensively reviewed in [5]) fall into this category. TrackMeNot, for instance, is a browser plugin that generates extra dummy queries to mislead a potential adversary that is trying to infer the interest profile of its users. A first problem with these solutions is the relatively large increase in the server's query load, which may not be tolerable by many online services. A second problem is that they do not consider personalization as a goal, thus significantly affecting the experience of services like web search.

In literature, we find four main types of obfuscation techniques that have been used to protect disclosure of a user's personal information, particularly location [26] and search queries [5], to online services. "Transformation-based techniques" which employ cryptographic transformations to hide a user's data and "progressive retrieval techniques" that iteratively retrieve candidate results from the server yield a high overhead for the client and cannot be implemented without the cooperation of the server. They have not been applied in the context of web search. Generalization techniques [26], [49] replace quasi-identifiers with less specific, but semantically consistent, values. These approaches are likely not to work for search as we have seen in §IV when comparing interest-based and URL-based profiles. Literature [18] in the web community has confirmed this result, although different opinions exist [49]. Bloom cookies adopt noise addition techniques (extensively described in [5]) which can be implemented at the client. Our baseline HYBRID is inspired by Plausibly Deniable Search (PDS) [31].

We present Bloom cookies in the context of web search. Extensive research has been done on search personalization. As shown in [18], personalized search approaches can be classified depending on whether information relating to a single user or a group of users is used to create the user profiles. Group-level personalization (e.g., collaborative filtering) requires the server side involvement. In this work, we focus on client-side solutions, and hence consider only person-level techniques. Person-level techniques typically mine user profiles at the granularity of user interests or URLs. There are two main approaches to achieve person-level re-ranking of search results [18]: URL-based profiles and interest-based profile. We implement both and compare their performance.

We are aware of only one other work that leverages Bloom filters as privacy structures [8]. The authors propose specialized techniques for setting bits in a Bloom filter with the goal of achieving deniability of the stored items. After implementing their techniques, we deemed them not suitable for our use case because they are too computationally expensive, they generate deterministic noise, and they still require the presence of a dictionary.

## VII. Limitations

**Small dataset.** We have evaluated Bloom cookies with the search logs of 1300 users. The number of users in a real online service can be several orders of magnitude more. The personalization quality for a certain user is not affected by other users in the system, and hence we expect our personalization results to be similar with a larger dataset. More users, however, are likely to improve the privacy of Bloom cookies. This is because with more users in the system, a user's noisy profile is more likely to "collide" with (i.e., look similar to) another user, making it difficult for the server to link a user's profile over time. This was confirmed by our experiments in §V-D. Moreover, as described in our threat model (see §II-B), using a small user population gives us a worst case scenario that takes into account cases in which a malicious server has other means to reduce the user population size (e.g., using geolocation, filtering by language, etc.)

**User feedback.** The privacy guarantee of Bloom cookies is a statistical one, rather than an absolute one. As our results show, Bloom cookies achieve unlinkability for a good fraction of the users, but there are users whose online activities and interests are so distinct that even with the amount of noise we added, they remained linkable across IP-sessions. We believe that it might be possible to provide feedback to those users so that they can decide to add more noise or use other measures

that provide better privacy at the cost of personalization. Our future work includes developing such feedback techniques.

**Longer term analysis.** Our evaluation assumes 2 consecutive 2-week periods of user activities. With longer user history this evaluation can scale to longer and more time periods.

**Applicability to other applications.** We focus on web search and use personalization techniques (and metrics) which are standard in the web literature. Commercial search engines might use more sophisticated but proprietary techniques. However, if personalization algorithms rearrange search results as the last step before returning them to users and if rearranging is performed solely on the presence of certain items in user profiles, our privacy-preserving approach will work in a straightforward way. This is true not only for web search, but also for services with a similar structure such as advertising (ranking of ads), online shopping (ranking of products), or personalized news delivery (ranking of news). It might even be possible for different services to share the data responsibly in a privacy preserving way, provided they are interested in similar information about the user, such as the top visited websites. However privacy-personalization analysis of such a scenario is out of scope of the current work.

**Tracking across services.** Unfortunately, tracking often occurs across services. For instance, Google or Microsoft can track users using web search, email, or voice calls. As discussed in our threat model (see §II-B), this scenario is out of scope for this paper, but this work sets the first step towards such a vision because IP-based tracking is unavoidable in client-server setups (unless anonymization proxies can be afforded). To address a multi-service scenario, it is first necessary to understand which information every service needs for personalization and then how it can be obfuscated. We chose web search also because of its relatively mature personalization, which is not the case for most services.

## VIII. Conclusions

Bloom cookies encode a user's profile in a compact and privacy-preserving way, but do not prevent online services from achieving personalization. Compared to profile generalization and noise addition techniques commonly employed in online privacy-preserving personalization systems, Bloom cookies provide a much better privacy, personalization and network efficiency tradeoff. Through the analysis of web search logs, we showed that profile generalization significantly hurts personalization and fails in providing reasonable unlinkability. Noise injection can address these problems, but comes with the cost of a high communication overhead and a noise dictionary which must be provided by a trusted third party. Bloom cookies leverage Bloom filters as a privacy-preserving structure to deliver similar (or better) personalization and unlinkability than noise injection, but with an order of magnitude lower communication cost and no noise dictionary dependencies.

## Acknowledgements

This work was partly supported by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA. We thank Doug Burger for initially suggesting Bloom filters for service personalization. We also thank Ryen White and Dan Liebling for help understanding web search personalization algorithms and processing search logs.

## References

[1] "Open Directory Project," http://dmoz.org/.

[2] G. Aggarwal, E. Bursztein, C. Jackson, and D. Boneh, "An Analysis of Private Browsing Modes in Modern Browsers," in *USENIX Security Symposium*, 2010, pp. 79–94.

[3] C. Ardagna, M. Cremonini, E. Damiani, S. De Capitani di Vimercati, and P. Samarati, "Location privacy protection through obfuscation-based techniques," in *Data and Applications Security XXI*, ser. LNCS. Springer Berlin Heidelberg, 2007, vol. 4602, pp. 47–60.

[4] M. Balakrishnan, I. Mohomed, and V. Ramasubramanian, "Where's that phone?: geolocating IP addresses on 3G networks," in *Proc. of IMC '09*, 2009, pp. 294–300.

[5] E. Balsa, C. Troncoso, and C. Diaz, "OB-PWS: Obfuscation-Based Private Web Search," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2012, pp. 491–505.

[6] P. N. Bennett, R. W. White, W. Chu, S. T. Dumais, P. Bailey, F. Borisyuk, and X. Cui, "Modeling the impact of short- and long-term behavior on search personalization," in *Proc. of SIGIR '12*, 2012, pp. 185–194.

[7] R. Bhagwan, S. Savage, and G. Voelker, "Understanding availability," in *Peer-to-Peer Systems II*, ser. LNCS. Springer Berlin Heidelberg, 2003, vol. 2735, pp. 256–267.

[8] G. Bianchi, L. Bracciale, and P. Loreti, "'Better Than Nothing' Privacy with Bloom Filters: To What Extent?" in *Privacy in Statistical Databases*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7556, pp. 348–363.

[9] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[10] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[11] M. Casado and M. J. Freedman, "Peering Through the Shroud: The Effect of Edge Opacity on Ip-based Client Identification," in *Proc. of NSDI '07*. USENIX Association, 2007, pp. 13–13.

[12] P. A. Chirita, W. Nejdl, R. Paiu, and C. Kohlschütter, "Using ODP metadata to personalize search," in *Proc. of SIGIR '05*, 2005, pp. 178–185.

[13] S. Claußand S. Schiffner, "Structuring Anonymity Metrics," in *Proc. of the 2nd ACM Workshop on Digital Identity Management*, ser. DIM '06, 2006, pp. 55–62.

[14] comScore, "The Myth of Static IP," Sept 2008, http://www.comscore.com/Insights/Blog/The_Myth_of_Static_IP.

[15] S. Cronen-Townsend and W. B. Croft, "Quantifying query ambiguity," *Proc. of HLT'02*, pp. 94–98, 2002.

[16] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: the second-generation onion router," in *Proc. of the 13th conference on USENIX Security Symposium - Volume 13*, 2004, pp. 21–21.

[17] J. Domingo-Ferrer, A. Solanas, and J. Castellà-Roca, "h(k)-private information retrieval from privacy-uncooperative queryable databases," *Online Information Review*, vol. 33, no. 4, pp. 720–744, 2009.

[18] Z. Dou, R. Song, and J.-R. Wen, "A large-scale evaluation and analysis of personalized search strategies," in *Proc. of WWW '07*, 2007, pp. 581–590.

[19] M. Franz, B. Meyer, and A. Pashalidis, "Attacking Unlinkability: The Importance of Context," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, 2007, vol. 4776, pp. 1–16.

[20] M. Fredrikson and B. Livshits, "RePriv: Re-imagining Content Personalization and In-browser Privacy," in *IEEE Symposium on Security and Privacy*, 2011, pp. 131–146.

[21] S. Gauch, J. Chaffee, and A. Pretschner, "Ontology-based personalized search and browsing," *Web Intelli. and Agent Sys.*, vol. 1, no. 3-4, pp. 219–234, Dec. 2003.

[22] D. Goldschlag, M. Reed, and P. Syverson, "Onion routing," *Commun. ACM*, vol. 42, no. 2, pp. 39–41, Feb. 1999.

[23] S. Guha, B. Cheng, and P. Francis, "Privad: Practical Privacy in Online Advertising," in *Proc. of NSDI '11*, Boston, MA, 2011.

[24] D. Howe and H. Nissenbaum, "TrackMeNot: Resisting Surveillance in Web Search," in *In Lessons from the Identity Trail: Anonymity, Privacy, and Identity in a Networked Society*, ser. Oxford: Oxford University Press, I. Kerr, C. Lucock, and V. Steeves, Eds., 2009, pp. 31–58.

[25] 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*, vol. 36, no. 2, pp. 207 – 227, 2000.

[26] C. S. Jensen, H. Lu, and M. Yiu, "Location privacy techniques in client-server architectures," in *Privacy in Location-Based Applications*, ser. Lecture Notes in Computer Science, C. Bettini, S. Jajodia, P. Samarati, and X. Wang, Eds., 2009, vol. 5599, pp. 31–58.

[27] A. Juels, "Targeted Advertising ... And Privacy Too," in *Proc. of the 2001 Conference on Topics in Cryptology: The Cryptographer's Track at RSA (CT-RSA 2001)*, 2001, pp. 408–424.

[28] H. Kido, Y. Yanagisawa, and T. Satoh, "An anonymous communication technique using dummies for location-based services," in *Proc. of ICPS '05*, July 2005, pp. 88–97.

[29] R. Krovetz and W. B. Croft, "Lexical ambiguity and information retrieval," *ACM Trans. Inf. Syst.*, vol. 10, no. 2, pp. 115–141, Apr. 1992.

[30] Z. Ma, G. Pant, and O. R. L. Sheng, "Interest-based personalized search," *ACM Trans. Inf. Syst.*, vol. 25, no. 1, Feb. 2007.

[31] M. Murugesan and C. Clifton, "Providing Privacy through Plausibly Deniable Search," in *Proc. of the SIAM International Conference on Data Mining (SDM '09)*, 2009, pp. 768–779.

[32] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *IEEE Security and Privacy*, 2013.

[33] M. S. Olivier, "Distributed proxies for browsing privacy: a simulation of flocks," in *Proc. of SAICSIT '05*, 2005, pp. 104–112.

[34] K. Purcell, J. Brenner, and L. Rainie, "Search Engine Use 2012," March 2012, http://pewinternet.org/Reports/2012/Search-Engine-Use-2012.aspx.

[35] D. Rebollo-Monedero and J. Forné, "Optimized query forgery for private information retrieval," *IEEE Transactions on Information Theory*, vol. 56, no. 9, pp. 4631–4642, 2010.

[36] B. Shapira, Y. Elovici, A. Meshiach, and T. Kuflik, "PRAW - A PRivAcy model for the Web," *J. Am. Soc. Inf. Sci. Technol.*, vol. 56, no. 2, pp. 159–172, Jan. 2005.

[37] X. Shen, B. Tan, and C. Zhai, "Implicit user modeling for personalized search," in *Proc. of CIKM '05*, 2005, pp. 824–831.

[38] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz, "Analysis of a very large web search engine query log," *SIGIR Forum*, vol. 33, no. 1, pp. 6–12, Sep. 1999.

[39] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, "Fast hash table lookup using extended bloom filter: An aid to network processing," in *Proc. of SIGCOMM '05*. ACM, 2005, pp. 181–192.

[40] R. Song, Z. Luo, J.-R. Wen, Y. Yu, and H.-W. Hon, "Identifying Ambiguous Queries in Web Search," in *Proc. of WWW '07*. ACM, 2007, pp. 1169–1170.

[41] D. Sontag, K. Collins-Thompson, P. N. Bennett, R. W. White, S. Dumais, and B. Billerbeck, "Probabilistic models for personalizing web search," in *Proc. of WSDM '12*, 2012, pp. 433–442.

[42] M. Speretta and S. Gauch, "Personalized Search Based on User Search Histories," in *Proc. of the IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*, 2005, pp. 622–628.

[43] L. Sweeney, "K-anonymity: A Model for Protecting Privacy," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, pp. 557–570, Oct. 2002.

[44] J. Teevan, S. T. Dumais, and E. Horvitz, "Personalizing search via automated analysis of interests and activities," in *Proc. of SIGIR '05*, 2005, pp. 449–456.

[45] J. Teevan, S. T. Dumais, and D. J. Liebling, "To personalize or not to personalize: modeling queries with variation in user intent," in *Proc. SIGIR '08*, 2008, pp. 163–170.

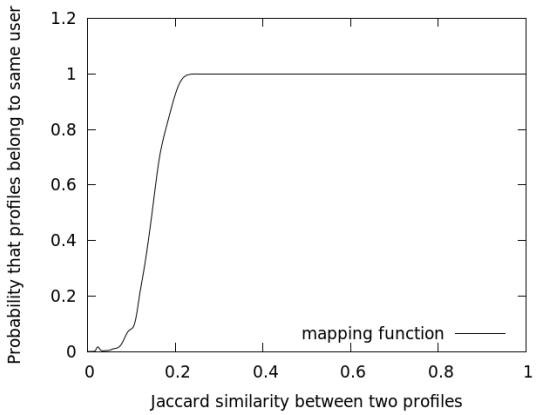



**Fig. 6: Linkability model trained over 300 users with exact profiles (URLs).**

[46] G. Tóth, Z. Hornák, and F. Vajda, "Measuring anonymity revisited," in *Proc. of the 9th Nordic Workshop on Secure IT Systems*, 2004, pp. 85–90.

[47] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas, "Adnostic: Privacy Preserving Targeted Advertising," in *Proc. of NDSS '10*, 2010.

[48] Y. Xie, F. Yu, K. Achan, E. Gillum, M. Goldszmidt, and T. Wobber, "How dynamic are IP addresses?" vol. 37, no. 4, pp. 301–312, 2007.

[49] Y. Xu, K. Wang, B. Zhang, and Z. Chen, "Privacy-enhancing Personalized Web Search," in *Proc. of WWW '07*, 2007, pp. 591–600.

[50] S. Ye, S. F. Wu, R. Pandey, and H. Chen, "Noise Injection for Search Privacy Protection," in *Proc. of IEEE CSE '09*, 2009, pp. 1–8.

## APPENDIX

**Linkability model.** As described in §III-B, the linkability model is essentially a function that maps the Jaccard similarity of a pair of user profiles to the probability of these profiles belonging to the same user. To calculate this mapping function, we take $n$ test users, and for each user we compute two profiles, one for the time period $T1$ and one for $T2$ (both 2-week long). Next, we calculate the Jaccard similarity for the $n^2$ profile pairs. We divide the entire range of possible similarities (varying from 0 to 1) into a fixed number (100) of buckets of equal size. For each bucket, we find how many of these $n^2$ profile pairs have similarities that lie in that particular range. From the fraction of these profile pairs that belong to the same user (known because the ground truth is available in this case), we calculate a conditional probability, i.e., the probability that a profile pair belongs to the same user given that the similarity of the profiles lies within a certain range.

Figure 6 shows such a mapping function with a Bezier interpolation[16] when using exact profiles (URLs). As expected, if the similarity is above a certain threshold, the probability that the profiles belong to same user becomes almost 1.

We calculate a mapping function also for generalized profiles and for each noisy profile we use in our evaluation. When noise is added to the profile, we first add the noise to the test users' profiles and then repeat the same process described above.

[16] http://en.wikipedia.org/wiki/Bezier_curve