University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, Heraklion, GR-70013, Greece

Privacy-Preserving Twitter Browsing through Obfuscation

Panagiotis Papadopoulos

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

Thesis Advisor: Prof. *Evangelos Markatos*

February 2014

University of Crete
Computer Science Department

Privacy-Preserving Twitter Browsing through Obfuscation

Thesis submitted by
Panagiotis Papadopoulos
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: Panagiotis Papadopoulos

Committee approvals:

Evangelos Markatos
Professor, Thesis Supervisor

Sotiris Ioannidis
Principal Researcher, Committee Member

Apostolos Traganitis
Professor, Committee Member

Departmental approval:

Angelos Bilas
Professor, Director of Graduate Studies

Heraklion, February 2014

Abstract

Over the past few years, microblogging social networking services have become a popular means of information sharing and communication. Although such services started as a convenient way of sharing small bits of information among friends, they are currently being used by artists, politicians, news channels, and information providers to easily communicate with their constituency. Even though following specific channels on a microblogging service enables users to receive interesting information in a timely manner, it may raise significant privacy concerns. For example, the microblogging service is able to observe all the channels that a particular user follows. This way, it can infer all the subjects a user might be interested in and generate a detailed profile of this user. This knowledge, being property of the microblogging service, can be used for a variety of purposes, most of which are usually beyond the control of the users.

To address these privacy concerns, we propose *k-subscription*: an obfuscation-based approach that enables users to follow privacy-sensitive channels, while, at the same time, making it difficult for the microblogging service to find out their actual interests. Our method relies on obfuscation: in addition to each privacy-sensitive channel, users are encouraged to randomly follow $k-1$ other channels they are not interested in. In this way (i) the actual interests of a user are hidden in random selections, and (ii) each user contributes in hiding the real interests of other users. Our analysis indicates that *k-subscription* makes it difficult for attackers to pinpoint a user's interests with significant confidence. We show that this confidence can be made predictably small by slightly adjusting k while adding a reasonably low overhead on the user's system.

Περίληψη

Τα τελευταία χρόνια, οι microblogging υπηρεσίες κοινωνικής δικτύωσης έχουν γίνει ένα δημοφιλές μέσο για την ανταλλαγή πληροφοριών και την επικοινωνία μεταξύ των χρηστών. Παρά το γεγονός ότι τέτοιες υπηρεσίες ξεκίνησαν ως ένας βολικός τρόπος για την ανταλλαγή σύντομης πληροφορίας μεταξύ φίλων, έφτασαν να χρησιμοποιούνται από καλλιτέχνες, πολιτικούς, κανάλια ειδήσεων, και διάφορους άλλους παρόχους πληροφοριών για να επικοινωνούν εύκολα με το κοινό τους. Οι χρήστες με το να ακολουθούν συγκεκριμένα κανάλια σε μια microblogging υπηρεσία έχουν τη δυνατότητα να λαμβάνουν ενδιαφέρουσες πληροφορίες άμεσα, το οποίο όμως μπορεί να προκαλέσει σημαντικά προβλήματα προστασίας της ιδιωτικής τους ζωής. Για παράδειγμα, μία microblogging υπηρεσία είναι σε θέση να καταγράφει όλα τα κανάλια που ένας συγκεκριμένος χρήστης ακολουθεί. Με αυτό τον τρόπο μπορεί να συνάγει όλα τα θέματα για τα οποία μπορεί ένας χρήστης να ενδιαφέρεται με αποτέλεσμα να μπορεί να δημιουργήσει ένα λεπτομερές προφίλ αυτού του χρήστη. Αυτή η γνώση, η οποία είναι ιδιοκτησία της microblogging υπηρεσίας, μπορεί να χρησιμοποιηθεί για διάφορους σκοπούς, οι περισσότεροι από τους οποίους είναι συνήθως πέρα από τον έλεγχο των χρηστών.

Για την αντιμετώπιση των θεμάτων που αφορούν προσωπικά δεδομένα προτείνουμε το *k-subscription*: μια προσέγγιση που προσφέρει κάλυψη στους χρήστες και τους επιτρέπει να ακολουθούν κανάλια που αφορούν ευαίσθητα θέματα χωρίς να υπάρχει διαρροή των προσωπικών τους δεδομένων. Την ίδια στιγμή, καθιστά δύσκολο για την microblogging υπηρεσία να μάθει τα πραγματικά ενδιαφέροντα των χρηστών. Η μέθοδός μας βασίζεται στην συγκάλυψη: πιο συγκεκριμένα οι χρήστες ενθαρρύνονται εκτός από το κανάλι που τους ενδιαφέρει, το οποίο αφορά ευαίσθητο θέμα, να ακολουθήσουν και άλλα $k-1$ τυχαία κανάλια. Με αυτόν τον τρόπο (α) τα πραγματικά ενδιαφέροντα των χρηστών κρύβονται ανάμεσα σε τυχαίες επιλογές, και επιπλέον(β) κάθε χρήστης βοηθά τους υπόλοιπους χρήστες να αποκρύψουν και αυτοί τα ενδιαφέροντά τους. Η ανάλυση μας δείχνει ότι το *k-subscription* καθιστά δύσκολο από την πλευρά των επιτιθέμενων, τον εντοπισμό των ενδιαφερόντων των χρηστών με βεβαιότητα. Τέλος, αποδεικνύουμε ότι η βεβαιότητα αυτή μπορεί να μειωθεί ελεγχόμενα προσαρμόζοντας απλά την παράμετρο k, προσθέτοντας παράλληλα μια αρκετά χαμηλή επιβάρυνση στο σύστημα του χρήστη .

Acknowledgements

I am grateful to my supervisor Prof. Evangelos Markatos for his guidance, for the chance to be a part of the Distributed Computing Systems Lab at FORTH-ICS and for introducing me to the notion of research. I also want to thank Antonis Papadogiannakis for all his contribution and effort put in this work. This thesis would not be possible without his advices and our countless talks. I need to express my appreciation to G. Vasiliadis, P. Garefalakis, A. Krithinakis for preserving the balance between computers and real life. Moreover, I would like to thank my colleagues S. Volanis, Z. Tzermias, L. Koromilas, T. Petsas, J. Polakis, L. Loutsis for making the working hours in the DCS lab fun and interesting. I want to express my sincere thanks to N. Skotis, A. Koutsouras, V. Ziaka, D. Patelis, D. Iwannidis, S. Ninidakis and all other friends for their support all these years. Moreover, I would like to thank Xristi who has always been there for me, supporting and helping me in all possible ways.

Last but not least, I am grateful to my parents, Evangelia and Manolis, and my sister Kelina for always being by my side, supporting and encouraging me through my studies and life.

This work was performed at **Distributed Computing Systems** laboratory, **Institute of Computer Science (ICS)**, **Foundation for Research and Technology – Hellas (FORTH)**, and is supported by the FP7 project SysSec, funded by the European Commission under Grant Agreement No. 257007.

Contents

List of Figures

Chapter 1

Introduction

Microblogging social networking services, such as Twitter[1], Tumblr[2], Identi.ca[3] and Fanfou[4], enable users to have timely access to all their information and entertainment needs. Through a publish-subscribe model, a user subscribes ("follows" in the language of microblogging) to a number of other users, or information providers in general ("channels"). These channels may correspond (i) to the user's friends, (ii) to artists the user is interested in, (iii) to politicians the user supports, (iv) to news channels, (v) to religious channels, (vi) to hospitals or doctors, and so on. When a new message is published in one of the channels a user follows, it appears on user's screen. This personalized information delivery, although useful, raises some privacy concerns. For example, if a user follows a particular politician, the service may be able to infer the user's political beliefs. If the user follows a religious channel, the service may be able to infer the user's religious preferences. If the user follows a channel about a particular health problem, the service may be able to infer that the user is interested in that health problem or may even be suffering from it.

As microblogging services become increasingly popular having hundreds of millions of users, the previously mentioned privacy concerns will become even more crucial in the future. Indeed, by providing a handy user interface and by co-locating all interests of a user into one convenient screen, microblogging services contain a huge amount of information about users' interests and needs. In this setting, there may be users that even though they wish to have access to timely information, they are not willing to disclose their personal preferences and interests to the microblogging service, and potentially to the corporations that may collaborate with it. One might argue that information provided by microblogging services is usually available even to users who are not logged in. This might be the case today for many channels, but it is not clear if it will remain so in the future. Several social networking services (such as Facebook and Google+) al-

[1] https://twitter.com
[2] https://tumblr.com
[3] https://identi.ca
[4] http://fanfou.com

ready limit the information available to non-logged in users, a trend we expect to influence microblogging services as well. Moreover, even if a user is able to access the information without being logged-in, the microblogging service might be able to correlate the content served with an IP address or a specific host.

To avoid being identified, users may log into the microblogging service using a pseudonym or a fake account. Although fake accounts might enable users not to disclose their real identification, it is not clear whether they are a good solution in the long run. Indeed, information from the user's IP address, third-party tracking cookies [18], or browser fingerprinting [12] may enable the microblogging service to pinpoint the identity of the user and trace the fake account to a real person. To make matters worse, if the user is logged into several services, then one might be able to correlate all these accounts through the use of the same IP address, web browser signature, or other common identifiers. Therefore, if the user employs a fake account in a microblogging service but a real account in another social network, it will probably be possible to correlate these two accounts. And finally, most social networks discourage users from giving fake data and creating fake accounts; some even ban users that provide only fake information. Overall, although a fake account might seem to be giving a sense of pseudonymity, it will not take long for a service to gather enough information to correlate a fake account with a real identity. A momentary lapse of vigilance is usually all it takes from the user to provide ample identifying information to the microblogging services. Another way to protect a user from being identified by a microblogging service is to use an anonymization service to access the Internet. For example, widely-used anonymous communication networks, such as Tor [10], are effective at hiding a user's real IP address, but are of little help when the user logs into a service such as a microblogging service. Indeed, in this case, the service will not be able to recognize a user's IP address, but it will have the user's identification since the user is logged in. To make matters worse, a potential technical (such as blocking of Tor nodes) or legal ban of Tor will make it very difficult, if not impossible, to use it anymore.

One might envision using both a fake account and Tor at the same time in order to have stronger anonymity. Unfortunately, this approach would still be subject to contamination from information gathered during different browsing sessions, such as browser fingerprints or cookies, which would alternate between anonymous and eponymous web browsing. To remedy this problem one might use (i) a fake account, (ii) an anonymization network, such as Tor, and (iii) a virtual machine per browsing session or destination web site, so as to limit cross-contamination. Although this triplet seems to provide a comfortable level of anonymity, its applicability and ease of use is questionable: it is not clear whether ordinary users will find it easy to install all the software needed to provide the desired level of anonymity. Also, modern devices, such as tablets and smartphones, which have rapidly penetrated the market, may not be able to install virtual machines.

Numerous approaches have been proposed to conceal users' activities when browsing the web, or interacting with social networks [4, 9, 13, 21]. However,

none of them can be adopted for microblogging services as the underlying threat model or technical implementation does not fit to the use case we address in this work. Most of the previously described approaches rely on the fact that in order to conceal one's interests we need to hide one's real identity. This work addresses a world where this is not easy, and may not even be possible. Indeed, a wide variety of existing web tracking mechanisms aided by increasing legal pressure on anonymity systems may lead to a world where anonymous web surfing would practically belong to the past. At this point we see two choices: (i) to believe that we will always be able to anonymously browse the web, and thus the massive losses of privacy we see in real life will never percolate to cyberspace, or (ii) to proactively develop privacy-preserving approaches for a world where it will be difficult, if not impossible, to hide one's real identity.

In this study, we explore the second choice and develop new obfuscation-based approaches to preserve privacy and conceal users' real interests. Whenever a user is interested in following an actual channel C_1, she is encouraged to follow $k-1$ additional channels acting as noise channels: C_1, C_2, C_3, ..., C_k. If the user follows all these channels, the microblogging service will not be sure which channel she is actually interested in. Moreover, if there is a plethora of users following the channel C_1 while they are not actually interested in it, the microblogging service will not be able to identify the users that are actually interested in C_1. By fine tuning this number of channels, users are able to achieve the level of privacy they are comfortable with.

Obfuscation itself is an age-old idea. It has been used in war to confuse radars detecting incoming bomber planes [32], in consumer organizations through super market card swapping to confuse marketers wanting to build customer profiles (e.g., Rob's Giant BonusCard Swap Meet[5]), and in web searches [14, 4] to hide the user's real interests from search engines. To the best of our knowledge, this is the first time that obfuscation is applied to provide privacy in the area of microblogging services. In cases where it is not possible to hide an event, such as an approaching war plane, a query to a search engine, and a channel followed in a microblogging service, obfuscation provides a reasonable mechanism to confuse the adversary to the point of not being able to distinguish real information from the added noise.

[5]http://epistolary.org/rob/bonuscard/

1.1 Contributions

In this thesis, we make the following main contributions:

1. We propose *k-subscription*: the first obfuscation-based approach to hide a user's interests in microblogging services. Our approach encourages users to follow $k-1$ noise channels in addition to every channel they want, so as to hide (i) their real interests in a set of k channels, and (ii) other users' interests in the microblogging service.
2. To quantify the effectiveness of our approach, we introduce a new notion: the *Disclosure Probability* P_C. This is the service's confidence that a user is interested in channel C.
3. We present an analytic evaluation of our approach and derive closed-form formulas for the disclosure probability. These formulas suggest that the disclosure probability can be made predictably small by fine-tuning the obfuscation level k.
4. We evaluate *k-subscription* in a more realistic scenario using simulations, which are based on models derived from a real-world dataset with sensitive channels from Twitter.
5. We implemented our system as a plug-in for the Chrome browser using Twitter as case study. We experimentally evaluate our prototype implementation and show that it has minimal bandwidth requirements and negligible latency to users' browsing experience.

1.2 Thesis Outline

The rest of this thesis is organized as follows. We present the design and analysis of *k-subscription*: our approach to obfuscate users' real interests in microblogging services. We study the anonymity provided by two different obfuscation strategies in an analytical way, and we evaluate the anonymity offered by *k-subscription* in a realistic scenario using simulations. To assess the practical feasibility and effectiveness of *k-subscription*, we have implemented an extension for the Chrome browser that enables privacy-preserving subscription to Twitter channels through obfuscation. Our experimental evaluation shows that the overhead introduced by *k-subscription* is reasonable in practice.

Chapter 2

Related Work

Privacy is an important aspect in online communications and thus this topic is an active research field. In the following, we discuss how our approach relates to previous work in this area.

2.1 Anonymous communications

One way to hide a user's accesses on the web is to use some anonymity network or an anonymization service [5, 10, 25]. Although such services can effectively hide a user's IP address, they cannot hide the user's identity if the user is logged into a microblogging service or if a subset of user's previous web accesses is known [24].

Recently, obfuscation has started to be used in the digital domain as well, to hide a user's digital tracks. Kido et al. [16] propose an anonymous communication technique for location-based services. Their technique is able to generate several false position data that is sent with the true information of the user to the service provider, in order to protect the user's location privacy.

2.2 Search engine query obfuscation

Howe and Nissenbaum [14] proposed *TrackMeNot*, a system designed to hide a user's real interest from a search engine. More specifically, for each real query submitted to the search engine, by a user, TrackMeNot also submits several other queries to confuse the search engine and introduce doubt for the user's real queries. TrackMeNot has been proven to be seminal work in the field and has led to the development of several other methods. For example, GooPIR [11] proposes an approach that is robust against timing attacks. For each real query, the user wants to submit, GooPIR constructs $k - 1$ other queries and submits all k of them at the same time to the search engine. This way, the search engine cannot construct a timing model on the user's real queries. Murugesan and Clifton [21] propose *Plausibly Deniable Search* (PDS) which gives users plausible deniability with respect to their search queries. Similar to GooPIR, each real query is accompanied

by $k-1$ other noise queries. Each real query, however, is also brought into a *canonical from* to prevent identifiability based on typos and/or grammar/syntax of the queries [22, 1]. Ye et al. [36] propose noise injection for search privacy protection. They give a lower bound for the amount of noise queries required for perfect privacy protection and provide the optimal protection given the number of noise queries.

Although the above systems are very effective at hiding *one* real query in a crowd of k queries, a determined adversary may be able to find a user's interests by studying successive sequences of queries. Indeed, if a user consistently generates authentic queries on a particular topic, but the $k-1$ "noise" queries added are on several different topics, then the adversary may easily find the user's real interests using clustering approaches. To protect against clustering attacks, PRAW [13] generates dummy queries on topics related to the topics the user is interested in.

Balsa et al. [4] compare previously proposed approaches and demonstrate their advantages and disadvantages. They show that many of them are vulnerable to various types of attacks, especially attacks that combine information from successive sequences of queries.

Our work shares ideas with the above works on search engine query obfuscation. However, it has a fundamental difference: in the field of search engine query obfuscation it is possible for some queries, especially the rare ones, to be submitted by only one user. Therefore, it is easy for the search engine to identify the users who submit rare queries and thus, to accurately find their interests. On the contrary, in *k-subscription* we always make sure that each channel, even the rare ones, is followed by lot of users. To put it simply: it is not how many "noise" channels a user follows – it is how many other users follow her channels of interest.

2.3 Hummingbird

Cristofaro et al. proposed *Hummingbird*, a system to provide privacy in Twitter [9]. The system assumes that a user (Alice) is interested in following a particular hashtag, e.g., from the New York Times (NYT). Hummingbird makes sure that neither Twitter nor NYT learn that Alice is interested in this hashtag. To achieve this, information providers (such as NYT) encrypt their tweets, and information consumers (such as Alice) are able to decrypt the tweets matching the hashtags they are interested in.

Although Hummingbird is effective at hiding the hashtags Alice is interested in, and seems related to our work, we see two main differences with our approach: (i) Hummingbird requires the *explicit* collaboration of the information provider (e.g., NYT) who should encrypt its tweets appropriately, and distribute keys so that Alice will be able to decrypt the tweets matching the hashtags she is interested in. In contrast, our system does not require any collaboration from the information providers: it is implemented on top of Twitter as it is today. (ii) Although a user in Hummingbird is able to hide the hashtag she is interested in, she cannot hide

the fact that she follows a particular channel (such as NYT). If, for example, Alice is interested in following a bankruptcy-related channel, Hummingbird will help Alice hide the hashtags in this channel, but she will not be able to hide the fact that she is interested in the bankruptcy channel. Our system is able to help Alice hide the fact that she is interested in this particular channel by making sure that she follows several other channels, and other people include this channel among their noise channels.

2.4 k-anonymity

Our work is similar to the concept of k-anonymity, which was first applied to anonymize patient data used for research in the medical field. This approach suggests that data should be anonymized in such a way so that any patient in the released dataset should be indistinguishable from at least $k-1$ other patients in the same dataset [28, 31]. Thus, if anyone having access to the data tries to discover whether a particular person A has disease B, there will always be $k-1$ other people "similar" to A.: that is, having the same address, belonging to the same group age, and so on.

To achieve k-anonymity, data are generalized so that any information that can uniquely identify a person will always point to at least k persons. For example, if the dataset contains only a single person living in a ZIP code area, k-anonymity removes some of the ZIP code digits so that there will be at least k people that share the same generalized ZIP code [23]. k-anonymity is frequently used together with l-diversity, which makes sure that all the patients in the same k-anonymity group do not suffer from the same disease [17].

Although *k-subscription* shares some ideas with k-anonymity, it has two fundamental differences: (i) the attack model is completely different. In k-anonymity, the owner of data tries to protect the user against third party adversaries. In our case, it is the user who tries to protect herself from the owner of data. (ii) k-anonymity assumes that the user cannot be identified by the adversary. In contrast, in our case, the user can easily be identified by the adversary, making it even more difficult to hide a user's data.

Chapter 3

System Design

In this section, we briefly outline our threat model and propose two different approaches capable to fortify a user's anonymity. Table 3.1 summarizes the notation we use throughout this section.

3.1 Threat Model

Let us now define the threat model we address. We assume the existence of a microblogging service where users are able to follow individual channels. A channel can be the account of a physical person, of an entity such as a corporation, of a news site, of a politician's office, and so on. Additionally, we assume that the microblogging service is capable of recording the users' interests by observing which channels each user follows. The information about the users' interests, which is property of the microblogging service, could be later sold to advertisers [27], and could be used for a variety of purposes, all of which are beyond the control of individual users [15]. We view this capability of the microblogging service as a potential concern for the users' privacy, and we would like to develop mechanisms that hide the users' real interests from the microblogging service.

In this work we assume an "honest but curious" microblogging service. In this aspect, the microblogging service may try to find the user's interests based on the channels the user is following, but it will not try to "cheat" by actively interfering with the process users are employing to protect their privacy, or try to gain more information than what a user is willing, or required, to give. For example, the microblogging service will not create fake channels or fake users in order to break the anonymity of ordinary users. We think that this "honest but curious" model is reasonable in practice, as popular microblogging services have a reputation they do not want to jeopardize by becoming hostile against their own users. Therefore, we expect such microblogging services to only try to passively gain knowledge based on data given by their users.

We also assume that when users *follow* channels, they act as consumers of information and refrain from interacting with any channel by posting information,

replying, retweeting, or sharing their interests in any other way. Indeed, if a user starts posting about a sensitive issue, it will be easy for the microblogging service to identify the user's interests.[1] We believe that most users want to find and consume information about a sensitive issue, and they will not take the risk of being identified by posting information about it. If some users would like to post, reply, or retweet anonymously about a sensitive issue, they may use *k-subscription* in combination with alternative solutions, such as #h00t [3] and Hummingbird [9].

3.2 Our Approach: k-subscription

Table 3.1 summarizes the notation we use throughout this section. Assume that user A is interested in following channel C, which deals with a sensitive issue, such as a medical condition. If user A follows only this channel, the microblogging service would easily figure out that A is interested in this medical issue. In this work we propose *k-subscription*: a system that makes sure that the microblogging service is not able to pinpoint A's interests with reasonable accuracy. To do so, *k-subscription* follows an obfuscation-based approach, which advocates that along with each channel C the user is interested to follow, she should also follow $k-1$ other channels (called "noise" channels). The number of noise channels is such that the microblogging service will not be able to determine A's interest with high probability, and will not be able to identify the actual set of users interested in each specific channel between the users who follow this channel as noise. All the noise channels are randomly chosen from a set S of "sensitive" channels. Note that A's real interests are also members of S. We have also implemented the proposed approach for the social network Twitter to demonstrate the practical viability.

In the rest of this section we describe a range of obfuscation algorithms that try to hide the fact that user A is interested in following channel C. Table 3.1 summarizes the notation we use throughout this section.

3.3 Uniform Sampling

When a user wants to follow channel C, *k-subscription* encourages the user to follow $k-1$ other channels as well (say C_1, C_2, ..., C_{k-1}). This way, the microblogging service will not know whether the user is actually interested in channel C or one of the C_1, C_2, ..., C_{k-1}. In our first algorithm, *k-subscription*-UNIF, these channels are chosen randomly with uniform probability from S. Algorithm 1 presents the pseudocode for *k-subscription*-UNIF.

[1] One might argue that in the spirit of *k-subscription*, a user may start posting about all subjects followed in all k channels, but it seems that it would be not very difficult for a smart microblogging social networking service to differentiate genuine posts which convey information from bogus posts whose purpose is to confuse the microblogging service and obfuscate the real post.

Table 3.1: Summary of Notation

Notation	Explanation
S	: Set of sensitive channels that can be followed
C	: Sensitive channel
U	: Number of all users in the system
U_C	: Number of users actually interested in channel C
U_{R_C}	: Number of users following channel C at random
p_C	: Popularity of channel C ($p_C = U_C/U$)
P_C	: Probability that a user following C is interested in C
N	: Number of sensitive channels a user is interested in
k	: Obfuscation level (per channel)

k-subscription-UNIF is a naive but powerful approach for obfuscation and we use it as a basic principle for our method. However, this approach leads to some practical problems. In case that not all channels enjoy the same popularity, then uniformly sampling from S may result in higher disclosure probability for the more popular channels. Thus, we discuss an improved version in the next section.

3.4 Proportional Sampling

A user following a popular channel (say C) along with several unpopular ones has a higher probability of being interested in C than in the rest of them. Capitalizing on this knowledge, the microblogging service has a better chance of finding those users who follow popular channels. To mitigate this issue, we propose *k-subscription*-PROP that sample channels from set S according to their popularity. Assume that U_C is the number of followers of channel C and $U_S = \sum_{\forall C \in S} U_C$ is the number of followers of all channels in S. Thus, instead of sampling all channels with probability $1/|S|$, we sample channel C with probability U_C/U_S. In Twitter, the popularity of a channel can be inferred by the number of users following the respective account. In other microblogging services similar metrics are available to determine the popularity of a channel. *k-subscription* with proportional sampling for adding noise does not affect the respective channel popularity.

3.5 Following Multiple Channels

So far we have dealt with users who are interested in obfuscating the fact that they are following one sensitive channel. Nevertheless, we expect that users may be interested in following more than one sensitive channel. Using *k-subscription*, users just need to select $k-1$ other noise channels to follow for *each* channel C they are interested in. Therefore, a user interested in following N channels will result in following $k \times N$ channels in total. However, it is very likely that a user will be interested in N sensitive channels that are *semantically related*. This case

ALGORITHM 1: *k-subscription*-UNIF: Choose noise channels uniformly from the set S

$F = \emptyset$; {initialize the set of channels to follow}
for $(i = 1\ ;\ i \leq k-1;\ i++)$ **do**
  C_i = randomly select a channel from set S ;
  $S = S \setminus C_i$; {remove C_i from S}
  $F = F \cup C_i$; {add C_i in the set of channels to follow}
end for
$F = F \cup C$; {add C in the set of channels to follow}
Follow all Channels in F in a random order;

may significantly increase the disclosure probability. Indeed, the microblogging service can easily find the correlated channels: it will get all the channels a user is following, classify them into semantic categories, and identify the sets of channels that are semantically related. If there is only one set of related channels, it is more probable that the user actually follows them, and the remaining unrelated channels are the selected noise. For example, a user may be interested to follow a channel dealing with recent news in a medical condition as well as another channel created by a support group for this same medical condition. These two channels will be highly related, while the rest of the $2k-2$ noise channels will probably not be so closely related. By constructing all pairs of channels and isolating the highly-related ones, the microblogging service will be able to find the interests of the user with a higher probability. In case the user is interested in more than two related channels, it may be even easier for the microblogging service to identify them among the unrelated noise channels.

One way to address this issue could be the following: whenever a user is interested in N related channels, the $(k-1) \times N$ noise channels could be selected in N-tuple groups, so that each N-tuple consists of N related noise channels. However, this approach has a certain limitation: the microblogging service and *k-subscription* may use different similarity metrics to identify related channels. For instance, the microblogging service may use a more fine-grained similarity metric to find out the actual related channels.

Fortunately, *k-subscription* is able to protect users' interests even when they are interested in multiple semantically related channels. Although a user will actually follow the set of N related sensitive channels she is interested in, which can be identified by the microblogging service, there will be a significant number of other users that also follow the same set of N related channels due to random noise channel selections, i.e., without being interested in them. This is due to the increased random selections when users are interested in multiple channels. Thus, the microblogging service will not be able to know which of the users following all these N related channels are actually interested in them.

Chapter 4

Analytical Evaluation

After having introduced different obfuscation algorithms, we now analyze the efficiency of our approach and provide closed form formulas for the disclosure probability. The disclosure probability P_C is the probability (as it can be calculated by the microblogging service) that a user who follows channel C is interested in C. In our analysis we assume that the microblogging service is able to infer each channel's popularity U_C by the number of its followers or other external information, the number of users U that have adopted *k-subscription*, the size of set S that is publicly released, and the value of k used by each user, e.g., as a user subscribes to these k channels in a short period.

4.1 Analysis of k-subscription-UNIF

A user, along with channel C follows $k-1$ other channels as well, hence the disclosure probability is $P_C = 1/k$. However, for large values of k (i.e., in cases where the user wants to add a lot of noise) the microblogging service has a more effective way to increase its certainty about the interest of a user in a particular channel C. It knows that the U_C users who are interested in channel C actually follow it. However, there are $U - U_C$ other users that are *not* interested in C who may have randomly included C among their noise channels. The probability of C being included in the set of channels followed randomly by a user interested in a channel *different* than C is bounded by $1-(1-1/|S|)^{k-1}$, as this user will select $k-1$ channels randomly from S, which also includes C. Therefore, the average number of the $U - U_C$ users not interested in C that will follow C randomly as noise (U_{R_C}) are less than $(U - U_C) \times (1-(1-1/|S|)^{k-1})$. So, the ratio of users following C who are really interested in C is less than:

$$\frac{U_C}{U_C + (U - U_C) \times (1-(1-1/|S|)^{k-1})}$$

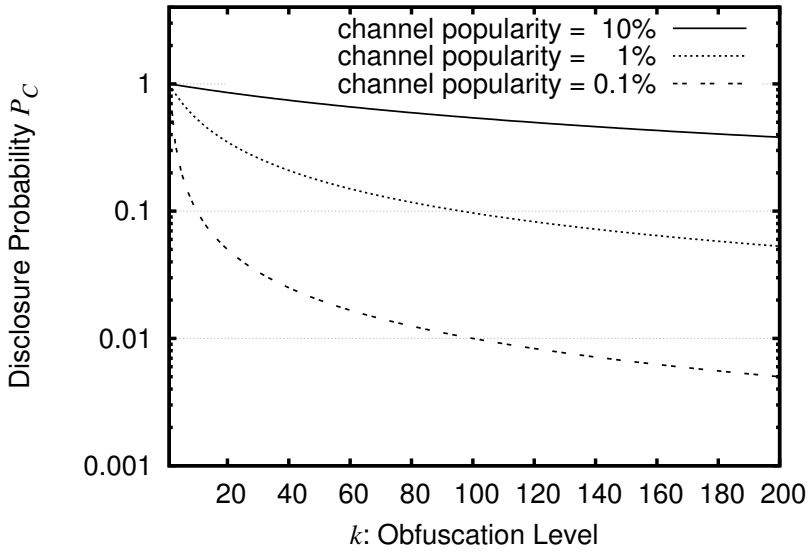


Figure 4.1: Disclosure Probability P_C of *k-subscription*-UNIF as a function of the obfuscation level k for different channel popularities.

Since the microblogging service does not know which U_C users are interested in the channel C, it can only assume that all users following C are interested in C. The probability of a user following C actually being interested in C (denoted as P_C) is given by:

$$\begin{aligned} P_C &< \max(1/k, \frac{U_C}{U_C + (U - U_C) \times (1 - (1 - 1/|S|)^{k-1})}) \Rightarrow \\ P_C &< \max(1/k, \frac{p_C}{p_C + (1 - p_C) \times (1 - (1 - 1/|S|)^{k-1})}) \end{aligned} \tag{4.1}$$

where p_C is the channel's popularity. We see that the total number of users U and the number of users U_C interested in channel C do not affect the disclosure probability. Instead, the channel's popularity p_C, the parameter k, and the total number of channels $|S|$, are the key factors that affect the disclosure probability.

4.1.1 Estimating the Disclosure Probability

In this subsection we are going to present how the disclosure probability P_C changes as a function of k (the level of obfuscation). First we arbitrarily fix $|S|$ to 1,000 channels. Figure 4.1 shows how the disclosure probability P_C changes as a function of k (the level of obfuscation). We see that when the popularity p_C of a channel is rather high (i.e., 10%), then it is difficult to obfuscate it with the *k-subscription*-UNIF approach. Indeed, when as many as 10% of the users are interested in channel C, then it would take a significant percentage of the rest 90% to include

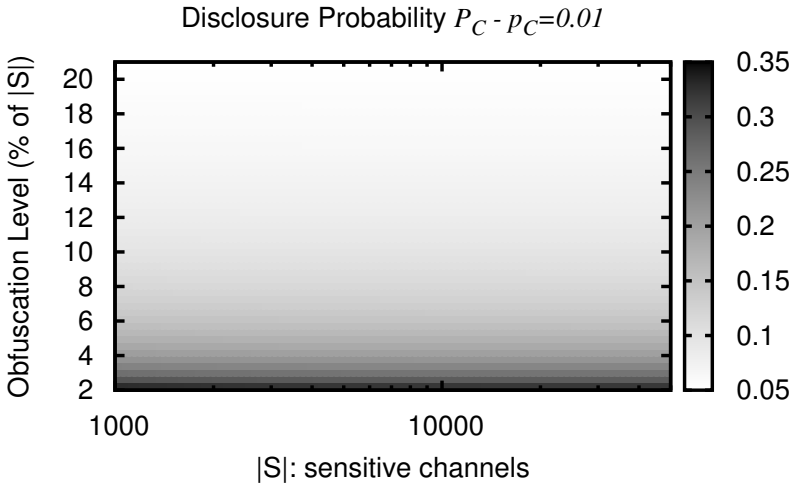


Figure 4.2: Disclosure Probability P_C of *k-subscription*-UNIF as a function of $|S|$ and the obfuscation level k, shown as a percentage of $|S|$, for channel popularity 1%.

channel C among their noise channels, which is very difficult to achieve. However, when popularity is around 1%, then it is much easier to obfuscate it using this approach. Indeed, for $k = 100$ the disclosure probability is as low as 0.1, which means that the microblogging service cannot state with confidence larger than 10% that a user A who follows channel C is really interested in C. Fortunately, when the popularity of C is even lower (i.e., around 0.1%), the disclosure probability becomes 0.01 for obfuscation levels as low as 100. That is, the microblogging service cannot say with confidence higher than 0.01 that a user who follows C is really interested in C.

In our next figure we explore how the disclosure probability changes as a function of the number of sensitive channels $|S|$ and the obfuscation level k. That is, if we double $|S|$ how should we increase k so as to have the same disclosure probability? Figure 4.2 shows a plot of the P_C as a function of $|S|$ and k. Note that the obfuscation level k in the y-axis is shown *not* as an absolute number but as a percentage of the number of sensitive channels. From this figure we clearly see that lines of the same color run horizontally. Horizontal lines mean that the value is the same for constant y (obfuscation level as a percentage of $|S|$). This implies that as long as the obfuscation level is a constant percentage of the size of the set of sensitive channels $|S|$, the disclosure probability remains constant. To put it simply, if we double the number of sensitive channels, we need to double the obfuscation level k (in absolute numbers) in order to keep the disclosure probability constant.

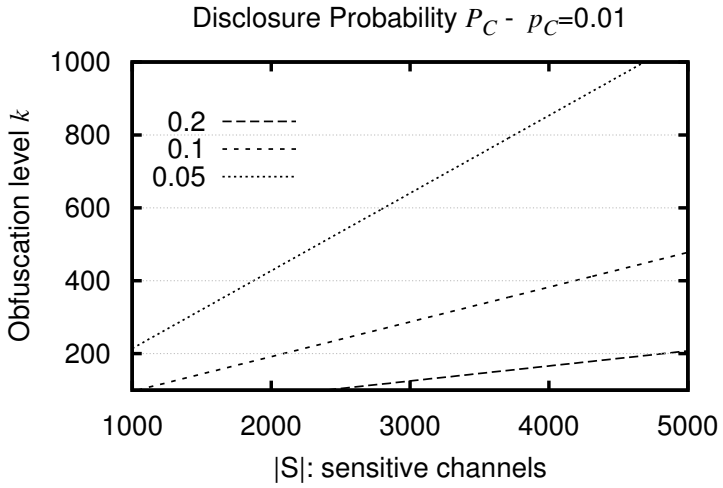


Figure 4.3: Disclosure Probability P_C of *k-subscription*-UNIF as a function of $|S|$ and the obfuscation level k, for channel popularity 1%.

To explore the relation between $|S|$ and k even further, Figure 4.3 shows the iso-probability contours of the disclosure probability P_C as a function of the number of sensitive channels (i.e., $|S|$ on x axis) and the obfuscation level (i.e., k on the y axis). We plot the contours for probabilities 0.05, 0.1, and 0.2. Interestingly, we see that the iso-probability contours appear as straight lines, clearly implying an almost linear relation between $|S|$ and k. That is, doubling $|S|$ would require a twice as high k in order to keep the probability of disclosure to the same level. Similarly, if we can afford to double the obfuscation level, we can provide the same disclosure probability for twice as many sensitive channels. Or, equivalently, if we are forced to half the obfuscation level, we can still provide the same disclosure probability but only for half as many sensitive channels. These results are very encouraging in the sense that we can still achieve the levels of P_C we are comfortable with, even when we are forced to use small obfuscation levels.

Figure 4.4 shows the impact that the number of sensitive channels $|S|$ and the channel popularity p_C have on the disclosure probability. The obfuscation level k is set to 100. We see that the iso-probability contours are almost straight anti-diagonal lines indicating an almost inversely proportional relation between $|S|$ and p_C.

4.1.2 Finding a Reasonable Size for S

We now analyze how large S should be and how we can influence it. One can easily see that the larger S is, the higher the disclosure probability will be (for

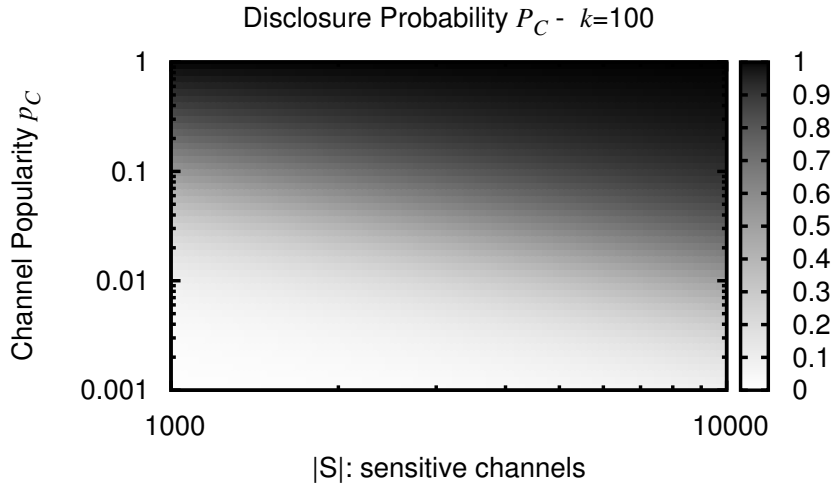


Figure 4.4: Disclosure Probability P_C of *k-subscription*-UNIF as a function of the size of S and channel popularity p_C. The obfuscation level k is set to 100.

a constant obfuscation level k). Therefore, we do not want the set S to be very large. On the other hand, a very small S would easily give away a user's true interests, and limit the users' choice for sensitive channels. Indeed, if S contains two channels, say C_1 and C_2, if a user follows C_1 (no matter whether she follows C_2 or not), the microblogging service will be able to conclude with probability $1/2$ that the user is interested in C_1. In the same spirit, if S contains n members, the microblogging service will be able to conclude with probability at least $1/n$ that the user is interested in the channel she follows. As a result, S should be large enough so that the probability $1/n$ is small enough, so as not to be useful for the microblogging service. In the absence of any other information, the microblogging service is able to conclude with probability U_C/U that a random user is interested in channel C. Therefore, the set S should be large enough so that $1/|S| < U_C/U$. We can estimate U_C from external sources, or based on the number of followers of channel C.

4.2 Analysis of k-subscription-PROP

Let us now derive closed formulas for *k-subscription*-PROP. The probability of C being included in the set of channels followed randomly by a user interested in a *different* channel than C is bounded by $1-(1-U_C/U)^{k-1}$, as this user chooses at random $k-1$ channels from S, and the probability of choosing C at each individual choice is U_C/U. The average number of users not interested in C who will follow C randomly as noise (U_{R_C}) is less than $(U-U_C)\times(1-(1-(U_C/U))^{k-1})$. So, the

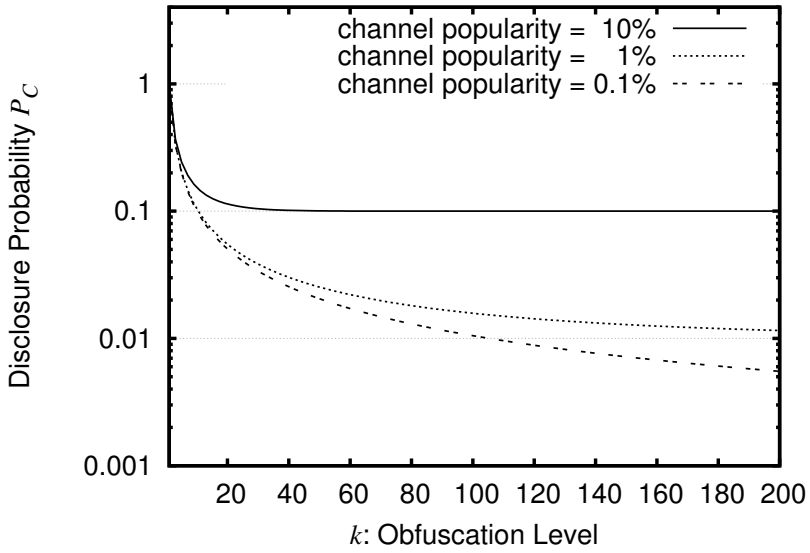


Figure 4.5: Disclosure Probability P_C of *k-subscription*-PROP as a function of the obfuscation level k. Sampling is proportional according to a channel's popularity.

ratio of users who follow C and are interested in C is less than

$$\frac{U_C}{U_C + (U - U_C) \times (1 - (1 - U_C/U)^{k-1})}$$

Since the microblogging service does not know which U_C users are interested in C, it can assume that all users following C are interested in C. The probability of a user following C actually being interested in C (i.e., the disclosure probability) is bounded as follows:

$$\begin{aligned} P_C &< \max(1/k, \frac{U_C}{U_C + (U - U_C) \times (1 - (1 - U_C/U)^{k-1})}) \Rightarrow \\ P_C &> \max(1/k, \frac{p_C}{p_C + (1 - p_C) \times (1 - (1 - p_C)^{k-1})}) \end{aligned} \tag{4.2}$$

where p_C is the channel's popularity. We observe that instead of the total number of users U and the number of users U_C that follow the channel C, the disclosure probability is affected only by channel's popularity p_C, the number of channels S, and obfuscation level k.

Figure 4.5 shows how disclosure probability changes with the level of obfuscation k. We see that our *k-subscription*-PROP approach is able to efficiently hide popular channels. Indeed, for a popularity of about 10% of the population of Twitter, it is able to reach a disclosure probability of 0.1 using only $k = 40$. When the popularity is 1%, even small obfuscation levels such as $k = 50$ can lead to disclosure probability as low as 0.02, which is very encouraging.

One can notice in Figure 4.5 that as k increases for 10% popularity, the disclosure probability tends to flatten out and in no case drops below 0.1. The reason is that for a channel with 10% popularity, the disclosure probability can never fall below 10% no matter how large the obfuscation level we use is. There is a simple explanation for this: without taking any channel-following information into account, the microblogging service knows that 10% of the population is interested in channel C. Hence, the microblogging service can safely assume that a user is interested in C with probability 0.1. Any further information should only be used by the microblogging service to increase this probability. Any obfuscation scheme, which will try to drag the microblogging service into a disclosure probability lower than 0.1, will just be abandoned by the microblogging service, which will fall back to its previous knowledge and safely assume that the probability a user is interested in C is at least 0.1.

4.3 Analysis for Multiple Channels

A user may want to follow more than one sensitive channel, which may be semantically related to each other. For simplicity, we assume that all users U are interested in exactly N sensitive channels from S. Each user that is actually interested to follow the N channels $C_1, ..., C_N$ will also follow $(k-1) \times N$ noise channels based on random choices from S.

Besides the $U_{C_1,...,C_N}$ users that are interested in these N channels, there will be some other users that will follow the same set of N channels without being interested in all of them, due to random noise channel selections. These users contribute to hide the actual interests of the $U_{C_1,...,C_N}$ users.

Since the microblogging service does not know the users who are actually interested in channels $C_1, ..., C_N$, it can only assume that all users following these channels are interested in them. The disclosure probability $P_{C_1,...,C_N}$ that a user following all these N channels is actually interested in them is equal to:

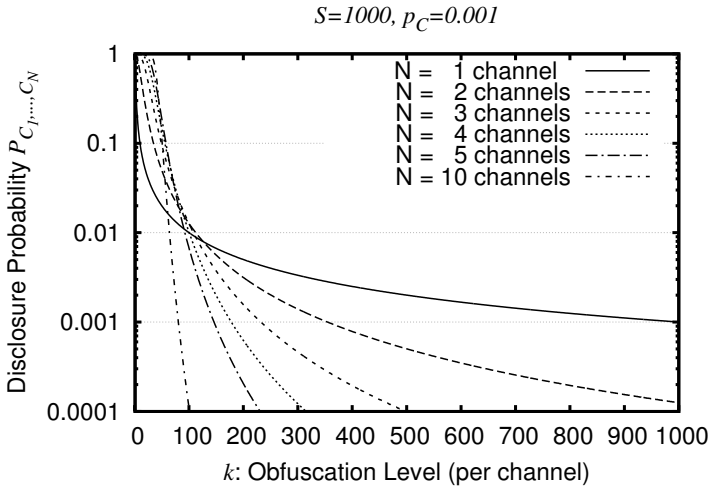
$$\frac{U_{C_1,...,C_N}}{U_{C_1,...,C_N} + (U - U_{C_1,...,C_N}) \times \binom{|S|-N}{(k-1)N-N} / \binom{|S|}{(k-1)N}} = \frac{p_{C_1,...,C_N}}{p_{C_1,...,C_N} + (1 - p_{C_1,...,C_N}) \times \binom{|S|-N}{(k-1)N-N} / \binom{|S|}{(k-1)N}} \tag{4.3}$$

where $\binom{|S|-N}{(k-1)N-N} / \binom{|S|}{(k-1)N}$ is the probability that a user selects randomly these N channels from the set S with $(k-1) \times N$ random choices when using the *k-subscription*-UNIF approach. We estimate this probability with a hypergeometric distribution, where all the successes N in the population S should be drawn with $(k-1) \times N$ attempts. Since we assume that all channels have the same popularity, the *k-subscription*-PROP approach has exactly the same behavior with *k-subscription*-UNIF in this analysis. $p_{C_1,...,C_N}$ is the popularity of the N-tuple of sensitive channels, i.e., the percentage of users actually interested in all these $C_1, ..., C_N$ channels.

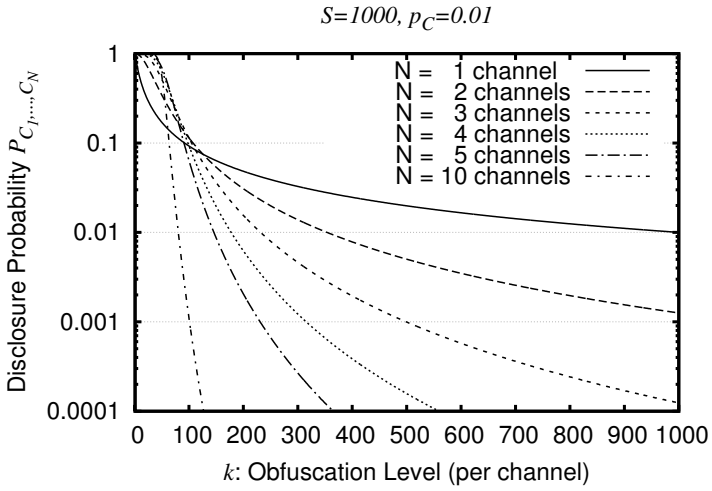
We want to explore how the disclosure probability changes with the number of channels N that a user may be interested in. We assume that the users interested in N channels are $U_{C_1,...,C_N} = U_C/N$, i.e., they are reduced by N times. Note that we assume a hyperbolic decrease of the users as N increases, instead of an exponential decrease, because we believe that these N channels will be probably semantically related. We set the size of S to $1,000$ and $2,000$ sensitive channels and we assume that all channels have the same popularity p_C, which is set to 0.1% and 1%.

Figure 4.6 shows the disclosure probability as a function of k for different values of N, ranging from 1 up to 10. We see that as the number of channels N increases, the disclosure probability is increased for low values of k, but it decreases significantly for higher values of k. The increase for low k values is because the users interested in N channels are reduced just by N times, following a hyperbolic growth, while the probability of randomly selecting these N channels for the rest of the users is reduced significantly by following a hypergeometric distribution. Thus, it is unlikely for the rest of the users to follow all these N channels at random with low k values. This means that for users interested in many sensitive channels we need to use a higher k value to achieve a low disclosure probability.

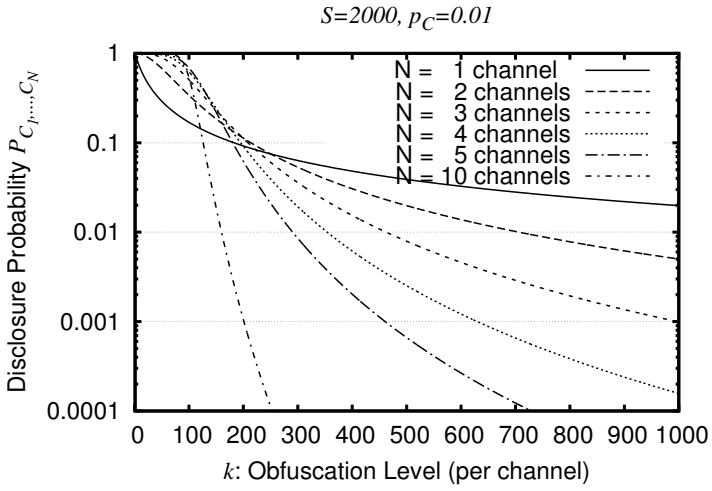
In contrast, for higher values of k, we see a significant reduction of the disclosure probability when users are interested in more channels. This is because the users interested in N sensitive channels follow $k \times N$ channels in total, so we have more random selections for higher N values. For instance, when $N = 5$ and $k = 200$, each user follows $1,000$ channels from S, i.e., all the existing channels in S when $|S| = 1,000$. The same happens in case of $N = 10$ and $k = 100$. When the size of S and channel popularity p_C increase, the disclosure probability increases respectively, according to Equation 4.3. However, a proper selection of a higher k value results in a much lower disclosure probability, as the users' interests can be efficiently hidden among the random selections of other users. Our experimental evaluation in Section 8 shows that the network bandwidth and latency when following few hundreds of noise channels are negligible, so our approach is able to protect the users' privacy even when they are interested in many sensitive channels that are probably semantically related.



(a) $|S|$=1,000, channel popularity=0.1%



(b) $|S|$=1,000, channel popularity=1%



(c) $|S|$=2,000, channel popularity=1%

Figure 4.6: Disclosure Probability as a function of the obfuscation level k when users are interested in one up to ten sensitive channels, for different size of S and channel popularity p_C.

Chapter 5

Simulation-based Evaluation

To evaluate *k-subscription* in a more realistic setup, where users are interested in a different number of sensitive channels, and sensitive channels have different popularities, we built a realistic Monte Carlo simulator. The simulator assigns a random popularity p_C to each channel following a similar distribution to real-world sensitive channels. First, each user randomly selects the number of channels N she is interested in following, based on a distribution similar to real-world users' selections. We assume that all N channels are semantically correlated. Then, the user selects these channels one-by-one at random, proportionally to channel's popularity. The noise selection is performed with *k-subscription*-PROP.

To simulate a realistic popularity distribution of sensitive channels, we selected a set S of $7,000$ sensitive channels using Twellow[1], a website that categorizes Twitter accounts according to their subject. The selected channels correspond to Twitter accounts dealing with health, political, religious, and other sensitive issues. We estimate the popularity of each channel based on its number of followers, i.e., U_C. Figure 5.1 shows the distribution of sensitive channel popularity in our dataset. We see that this distribution can be approximated very well using a power law with exponential cutoff model. We use this approximation in the simulator to assign a popularity p_C in each channel. We also see that only a small percentage of the sensitive channels exhibit relatively high popularity, which increases the disclosure probability. In contrast, the majority of sensitive channels have low popularity, which results in low disclosure probability even for low values of k.

To simulate a realistic distribution of the number of sensitive channels N each user is interested in, we used the same real-word dataset of sensitive channels. From the total $7,000$ channels, we used the Twitter API [2] to collect the user IDs of the followers of 500 sensitive channels related to disability issues, and we measured the number of occurrences of each user ID.

[1] http://www.twellow.com/categories/

[2] https://dev.twitter.com/docs/rate-limiting/1.1

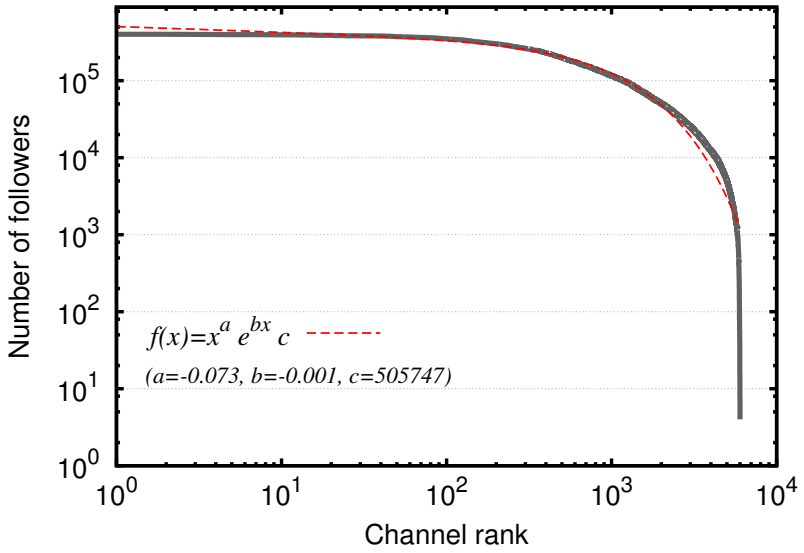


Figure 5.1: Distribution of the sensitive channels popularity.

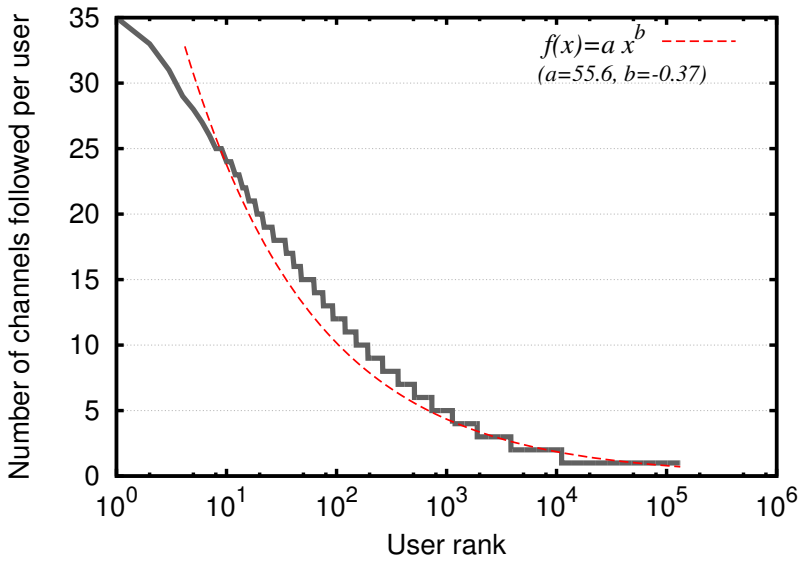


Figure 5.2: Distribution of the number of sensitive channels followed by a user.

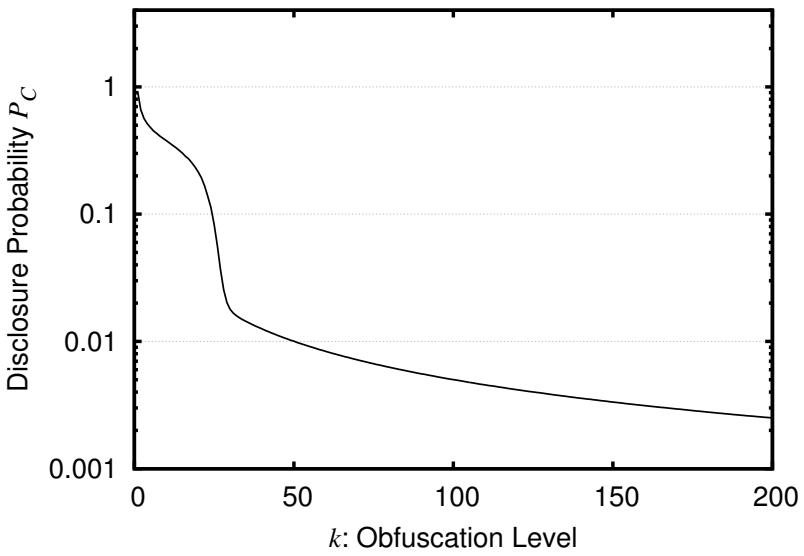


Figure 5.3: Disclosure probability as a function of obfuscation level k using realistic simulations.

This is the number of channels belonging in S that each user in our dataset follows. In this analysis we found more than $530,000$ unique users. Figure 5.2 shows the respective distribution, which is approximated with a typical power law function. This approximation is used in our simulations for realistic user selections. Also, we observe that only 0.85% of the users follow more than 4 sensitive channels, while 91.65% of the users follow just one sensitive channel.

The simulator keeps two counters per each channel: the number of users that (i) select this channel as actual interest (U_C), and (ii) select this channel as noise (U_{R_C}). Before exiting, the simulator reports the disclosure probability per channel, which is $max(1/k, U_C/(U_C + U_{R_C})$. Additionally, it keeps two lists per user: (i) the channels she is interested in (C_i), and (ii) the channels she selects as noise (C_n). This way, the simulator reports the disclosure probability per user, based on the set of sensitive channels the user is interested in (C_i). This probability is equal to $max(1/k, U_{C_i}/(U_{C_i} + U_{R_{C_i}}))$, where U_{C_i} the number of users interested in C_i and $U_{R_{C_i}}$ the number of users that C_i is included within their C_n. This is because we assume that all channels in C_i are semantically correlated. Among all the disclosure probabilities reported per user and channel, the simulator reports the overall average and maximum disclosure probability. We repeat each simulation for 100 times and we use the average values.

We set $|S|$ to 1,000 channels, U to 1,000,000 users, and we vary k from 1 to 200. Figure 5.3 shows the average disclosure probability reported by the simulator as a function of k. We see that *k-subscription* achieves a low average disclosure probability over all channels and users, which decreases rapidly with k. However,

we see that the average disclosure probability is increased for low values of k up to $k = 30$. This is because there are users interested in an N-tuple that no other user has selected among her random noise choices, especially for large values of N. However, as k increases, we see that an increasing number of users tend to select a significant percentage of the channels in S as random choices, e.g., users with large value of N. As these users follow most of the channels in S, they tend to hide the actual users' interests, even for large and rare N-tuples, reducing effectively the disclosure probability. In order to protect Twitter users' personal data, no names or usernames were included in our set S but only their userID serial No.

Chapter 6

Sensitive Channels

In this section we describe how the set S of sensitive channels is formed in *k-subscription*. We first present a technique for dynamically updating S with new sensitive channels using a sensitive channels' index, a mechanism used upon anonymous user requests for channels they are interested to follow and do not belong to S yet. Since the use of an updated S is a necessary factor for the efficient functionality of our system, each user of *k-subscription* periodically downloads the updated set of sensitive channels S from the aforementioned indexing service. Furthermore, we discuss defensive mechanisms against malicious users aiming at polluting S with fake channels in order to increase the disclosure probability. We show that *k-subscription* indexing service is able to filter out malicious requests for new channels and update S only with actual sensitive channels from benign users. Finally, we discuss how the channel indexing service may control the size of S to keep it smaller than a desired threshold.

6.1 Updating the Set of Sensitive Channels

To be effective, our system needs to make sure that all users share the same set S of sensitive channels. This is due to the fact that users' random selections of "noise" channels from S contribute to hide the real interests of other users, which are also channels belonging to S. Initially, S can contain a set of commonly used sensitive channels from several different categories. But, since new sensitive channels may arise, or users may want to follow a sensitive channel that is not included in S, there is a need to update the set of sensitive channels dynamically based on user needs.

To accommodate this necessity, we propose a Web-based indexing service that hosts the latest version of the set S with sensitive channels. *k-subscription*, which runs transparently at the client side as a Web browser plug-in, periodically communicates with this service to receive the updated set of sensitive channels. This way, each user is equipped with the latest version of S, that includes any newly added sensitive channels.

Moreover, this indexing service is able to accept user requests for new channels to be included in S. The user requests for adding a new channel in S are performed transparently and anonymously from the *k-subscription* browser plugin. When a user wants to follow a sensitive channel, *k-subscription* intercepts the "follow" request and checks whether this channel already exists in S. If so, the *k-subscription* algorithm runs as described in Section 3 by selecting and following $k-1$ noise channels. If the channel does not exist in S, the *k-subscription* plug-in communicates anonymously with the indexing service counterpart by sending a request with the new channel to be added. It is important that this communication, between user and *k-subscription* indexing service does not reveal the identity or the IP address of the user, so that the service will not know which user asked for which channel, and hence the actual user interests. To achieve anonymous communication, our prototype implementation of the plug-in uses Tor when sending requests towards the sensitive channel index. In addition, upon sending a request to include a new channel in S, the *k-subscription* plugin waits for a reasonable period of time before actually following this channel (and noise channels), so that several other users will have downloaded the updated S that includes the newly added channel. Upon accepting a request for a new channel, the sensitive channel indexing service adds the channel to S in the most relative category, using a classification algorithm based on term extraction. In addition, some of the existing channels in S may close or become inactive. The Web-based indexing service counterpart of *k-subscription* is also responsible to detect and remove these closed or inactive channels from S.

6.2 Defending Against Malicious Requests

It is theoretically possible that some malicious users may try to attack the proposed system by inserting to S non-sensitive channels, or even completely fake channels. A possible incentive of this attack is to increase the disclosure probability by adding a large number of channels in S, as we discussed in Section 4. This way, by filtering out the fake channels, an adversary can identify easier the actual sensitive channels along with the users interested in them. For instance, a set of colluding malicious users wanting to identify the legitimate users of our system, will insert batches of non-sensitive channels to S. This will result in overflowing our system with non-sensitive channels and consequently, the legitimate users will not be able to remain hidden in the crowd any longer.

To defend against the malicious actions discussed above, we protect our system by applying various security levels automatically when a user requests to add a new sensitive channel. As a first line of defense, the user needs to solve a set of CAPTCHAs before adding a new channel. The assumption here is that it is harder for automated methods (i.e., computer bots on Twitter [30], [35]), than it is for a real user to solve this type of challenges. We assume that this practice is relatively secure, given the fact that web pages like Google, Facebook, and Twitter, leverage it to protect themselves against adversaries that create accounts automatically.

Although there are a few published studies proposing solutions on how to break a CAPTCHA, mostly by using OCR algorithms [20], none of them is able to provide a concrete formula that can work in every single situation [7], [19], [29], [34], [33]. The probability is getting even lower if the proposed methods need to solve several CAPTCHAs sequentially. One can claim that this tactic leads to a user-unfriendly system but we believe that privacy is a more important aspect for our system than usability. Besides that, this is an action performed rarely by a user, only when she needs to follow a sensitive channel that is not included in our set.

However, the adversary could be a human, and not a computer program. In this case the previous challenge could effortlessly be surpassed. Therefore, we need another shielding approach as a second level of security. This is the *sensitivity* of the submitted channel. When a new channel is requested, we validate automatically that this channel is indeed a sensitive channel. The method we use here consists of two parts: (i) extraction of the channel's terminology, and (ii) evaluation of the extracted terminology for its sensitivity. For the first part, we use the *term extraction* technique. With this technique we extract the significant terms from a given channel in order to measure its sensitivity, and then categorize it. We obtain the noteworthy terminology by leveraging the Yahoo!'s Term Extraction API [1]. To evaluate the channel's sensitivity, we compare the extracted terms with a list of sensitive idioms. Those idioms are divided in semantic categories. For our case study we selected 43 categories covering health problems, political beliefs and religious preferences. If the comparison can reach a specific threshold, this channel is categorized as *sensitive* and is added to S, otherwise the procedure terminates with the channel not added to S.

An attacker, however, could try to bypass our protection mechanisms by adding sensitive content to a fake channel, and periodically posting tweets on it with content to this sensitive issue. Indeed, this way the fake channel would look similar to a sensitive channel and match the sensitive terms we defined. Thus, we use two more countermeasures: validating *channel's activity* and *channel's audience*. Given the fact that the submitted channel contains sensitive content, we need to evaluate whether this is *active* or not. We classify a channel as active by monitor the behavior of the channel over time. More precisely, we retrieve the content of the channel and calculate how often it is refreshed with new posts. In case that there is a period of inactivity, we categorize this channel as *dead*. The threshold for the inactive period is usually a few weeks, but the exact time-period is determined according to the distribution of channel's posts over time. This procedure is performed periodically by our system, and its main goal is to remove the *dead* channels from the existing sensitive set.

As a last line of defense, we authenticate the *audience* of the channel. We could easily identify a submitted channel that looks sensitive but has zero or very small number of followers, and exclude it from S. Also, it is not clear that the followers are actually humans, and not automated computer programs, known as bots, thus

[1] http://developer.yahoo.com/search/content/V1/termExtraction.html

they need to be examined as well. The criteria to distinguish the bots are listed as follows. First, we compare the amount of *followers* to the amount of *following* on these accounts. It is a solid indicator for this account to be bot if the above ratio is extremely low. Second, we compare the number of posts coming from an API, against the ones that are coming from web, or mobile. Typically, the automated methods, such as Twitter's API, are not widespread to normal users; consequently, this is a sign for indexing an account as suspicious. Third, the duplicate posts can also raise suspicion for a computer program. Fourth, the overflowing existence of spam, or malicious URLs in tweets, are factors we take into consideration. Fifth, many bots use to post tweets with unrelated links. An example is when the tweet description is completely irrelevant with the topic of the redirected web pages. Sixth, the automated generation of posts, like automatic updates of blog entries or RSS feeds, is another barometer we use. Seventh, many bots have a propensity to follow each other. Such behavior can be easily recognized and these accounts can effortlessly be classified by our system. Those criteria, along with the automated system for categorizing the Twitter users in humans or bots, have been extendedly analyzed in the past [8], [6]. Overall, if the majority of the followers are humans and not bots, the channel is added to the set of sensitive channels S.

6.3 Controlling the Size of Sensitive Channels Set

Besides malicious requests for fake channels, the size of S may increase normally even with only actual sensitive channels. Indeed, as users add new sensitive channels, S will continue to get larger over time. In Section 4 we concluded that when the size of S increases more than a threshold value, the disclosure probability starts to increase as well. To address this issue, the sensitive channel index is responsible to keep S in the desirable size. When S exceeds the size threshold, the channel index automatically removes from S the less important channels, i.e., less popular channels, less active channels, or older channels (based on the date they were inserted to S). This way the most important channels, that are more popular, active, or recently added to S, will remain in the set of sensitive channels. Note that if a user wants to follow a channel that has been previously removed from S, she can always send an anonymous request to the indexing service to add this channel back to S.

Chapter 7

Implementation

To evaluate the feasibility and efficiency of *k-subscription* we have implemented a Twitter extension for the popular Google Chrome web browser. The extension uses Twitter API v1.1 [1] and complies with the REST API Rate Limit. It is developed using JavaScript and JQuery, Json2, OAuth and Secure Hash Algorithm (SHA-1) libraries.

Figure 7.1 shows the overall operation of *k-subscription* extension. Upon installation, users can follow Twitter accounts in exactly the same way, through Twitter's web interface or "Follow me" buttons in third-party web pages. To enhance user's privacy, *k-subscription* intercepts all follow requests and checks whether they correspond to sensitive channels contained in S. If so, the extension transparently subscribes the user to $k-1$ additional "noise" channels from S according to the *k-subscription*-PROP algorithm presented in Section 3.4. These channels remain hidden and the user never interacts with them, in order to provide exactly the same Twitter browsing experience as before. To succeed this, the extension keeps a list of all "noise" channels and dynamically filters out the unsolicited tweets of these channels from user's feed, similarly to existing customization extensions such as Open Tweet Filter [26]. Other affected information, such as the number of channels followed, is adjusted appropriately by excluding the effect of the "noise" channels. Note that the same concept applies to other microblogging services as well; it is just a matter of extending our prototype.

At the first run, the extension downloads the set S of sensitive channels used for obscuring user's selections. The set includes information about each channel and the number of its followers to improve "noise" selection. The user can interfere with "noise" selection by proposing channels with predefined features such as language, country *etc.*

The users can disable the effect of *k-subscription* on a follow request when they are sure that they want to follow a non-sensitive channel. The users are also able to change some options like k parameter, refresh intervals, *etc.* Moreover, when a user unfollows a sensitive channel, the extension will also remove the "noise" added

[1] https://dev.twitter.com/docs/twitter-libraries

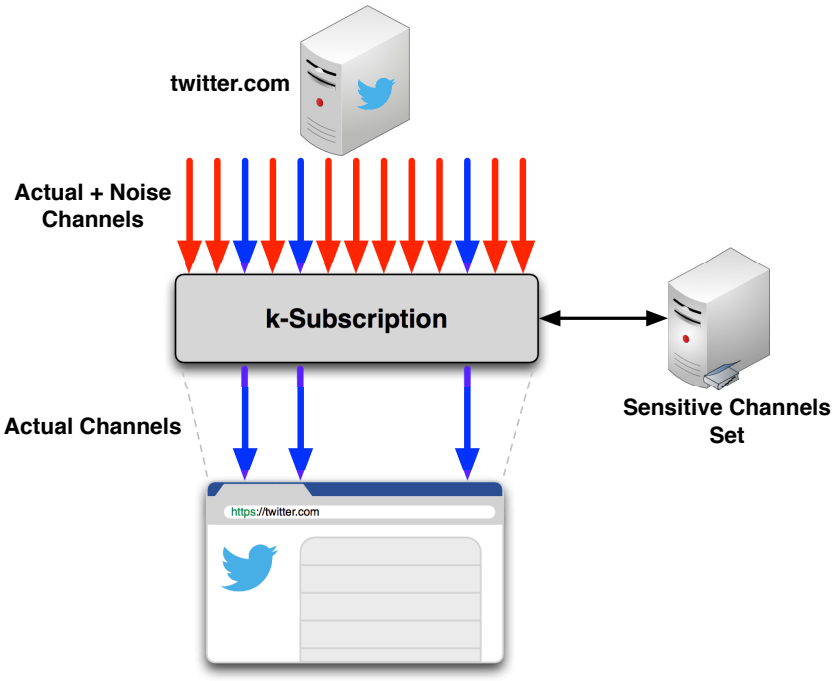


Figure 7.1: Overall operation of the *k-subscription* browser extension for Twitter. Whenever a user follows a new sensitive channel, *k-subscription* transparently follows additional "noise" channels and removes the "noise" from user's feed.

when the sensitive channel was followed. On the other hand, the extension checks periodically for sensitive channels the user has unfollowed from a different browser or machine. If such a case occurs, the extension considers both the channel and its noise as garbage and starts to unfollow them transparently.

We envision that the set of sensitive channels S along with the project in general would be maintained based on what we describe in Section 6. This maintenance can be done by the broader community of users and/or Non-Governmental Organizations (NGO) that have a specific view towards protecting privacy. Communities such as the Electronic Frontier Foundation have showed that they have the willingness and capacity to support similar initiatives. Hence, S can be seeded by an initial set of sensitive channels (such as medical- and political-related), and further improved through human intervention and participation of the community. Similar privacy-concerned projects, such as Tor, enlist the help of volunteers to maintain and improve its networks of routers. We expect that similar approaches can be applied for *k-subscription* as well.

Although *k-subscription* is effective at hiding the channels a user is interested in, a microblogging service may be able to find the user's real preferences by collaborating with URL shortening services. Most microblogging services limit the number of characters per each message (tweet). Therefore, several messages

include a URL pointing to more information. To reduce the number of characters taken by the URL, microblogging services provide (or collaborate with) a URL shortening service [2]. Users who click on a short URL are initially directed to the URL shortening service (which may be operated by the microblogging service, such as `t.co` in the case of Twitter) and then redirected to the actual URL. By monitoring which short URLs a user clicks, the microblogging service can learn the user's interests. We solve this problem within the *k-subscription* browser extension: whenever a user clicks on a short URL, *k-subscription* opens all short URLs posted in every channel the user follows. These URLs are resolved to the final destination URLs without the browser receiving a single byte from the targeted web pages. Then, the extension serves to the user only the actually requested URL. This way, the microblogging service will not be able to see which URLs a user clicks, as *k-subscription* transparently opens all of them.

Chapter 8

Experimental Evaluation

In this section we evaluate our implementation in terms of performance and effectiveness. Furthermore, we compare *k-subscription* implementation with a different widely used anonymization service, such as Tor when it is accompanied with a fake account.

8.1 Environment and Dataset

For all our experiments we used a PC equipped with an Intel Core 2 Duo Processor E8400 with 6MB L2 cache, 4GB RAM, and an Intel 82567 1GbE network interface. To populate the set S with sensitive channels we used Twellow a web site that categorizes Twitter accounts according to their subject. We created a set S with $7,000$ Twitter accounts dealing with health issues, political beliefs, abuses, religious preferences and more.

8.2 Adding Channels

In this first set of experiments we set out to explore the delay imposed by *k-subscription* when adding several noise channels in order to follow and hide a sensitive channel the user is interested in. Figure 8.1 shows how much time it takes to follow a sensitive channel with *k-subscription* as a function of k, i.e., when also following $k-1$ noise channels. We repeated our measurement for each k 100 times with random choices, and we report the average values. We see that the latency is an almost linear function of k, as expected. Fortunately, the time to follow several tens of channels is not significant. Indeed, it takes a little more than 1 minute to follow 100 channels and around 2 minutes to follow 200 channels. Since this operation is done only once, i.e., when a user wants to follow a sensitive channel, we believe that it does not add any significant overhead. Moreover, this operation runs as a background process, so it does not affect the user's experience.

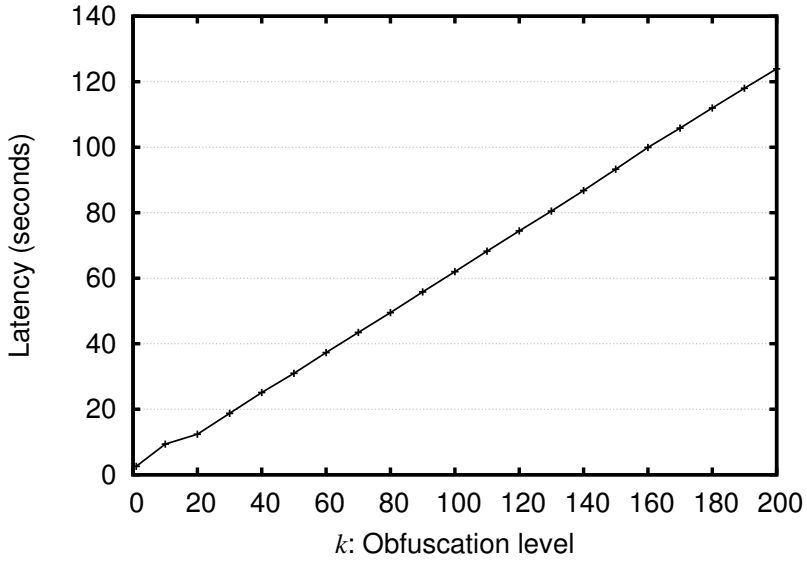


Figure 8.1: Time to follow a sensitive channel as a function of obfuscation level k.

8.3 How Much Does the Noise Cost?

In our next experiment we tried to quantify the additional amount of traffic the noise channels generate. To do so, we measured the total number of tweets generated by all channels, divided by each channel's lifetime and found the average number of tweets per channel per unit of time. The CDF of this function is shown in Figure 8.2. We see that the median channel (y=50%) generates less than one tweet (actually 0.25 tweets) per hour while 93% of the channels generate less than two tweets per hour. Overall, we see that the extra traffic generated by the noise channels should be very small. Even adding 100 noise channels generates no more than 25 tweets per hour, a negligible amount of traffic by most standards.

The reader will notice that the maximum posting rate that we have observed is about 6 posts per hour (averaged over the entire lifetime of the channel). Published statistics e.g., Twitaholic [1] suggest that the most prolific twitter accounts post as much as one tweet update per minute. Such accounts usually belong to news stations or even to automated programs (bots). Given that each tweet corresponds to just few hundred of bytes transferred over the network, even in such cases the resulting network overhead will be relatively low.

[1] http://twitaholic.com/top100/updates/

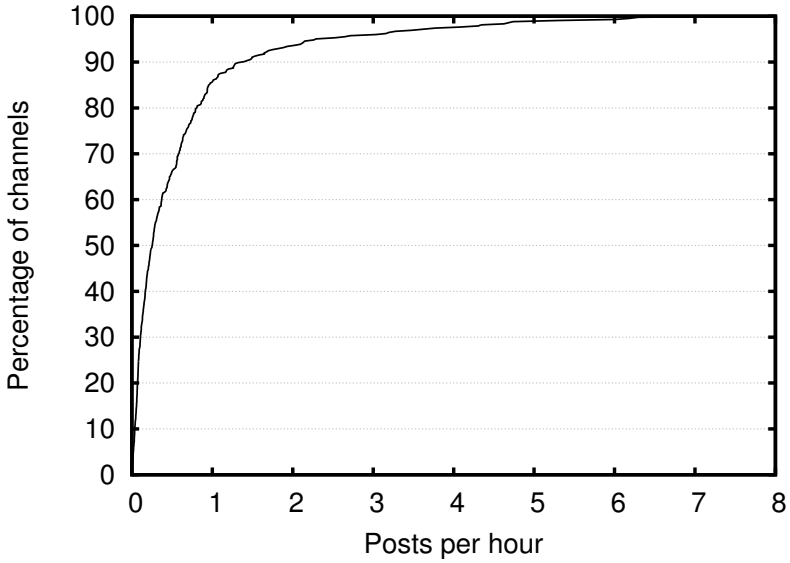


Figure 8.2: Number of tweets posted per channel per hour.

8.4 Bandwidth Consumption

When a user follows k channels for each subscription, she downloads roughly k times more information than she actually needs. However, we would like to see if the bandwidth needed for these downloads can be sustained by a home DSL Internet connection or not, and the respective network overhead in terms of used bandwidth. A user interested in N sensitive channels will receive tweets from $N \times k$ channels. The network overhead will be the same when a user is interested in one channel with N times higher k value. Thus, we evaluate our system while varying only the k parameter, assuming a user interested in a single channel. Figure 8.3 shows the traffic load generated by our implementation over a 30-minute period for a user following one sensitive channel with $k = 100$, $k = 50$, and $k = 1$ (i.e., without using *k-subscription*). We notice that the bandwidth consumption even in case of $k = 100$ is reasonably low, usually less than 1.5 Kbps. We see that even in case of the vanilla system (see $k = 1$) the bandwidth consumption is not significantly lower than in high values of k. In all cases it is usually between 0.5 and 1.0 Kbps. By manually inspecting the traffic we found that most of the bandwidth is used to download information like images, trends and recommendations, which does not depend on the value of k. The bandwidth used for downloading the actual tweets, which increase with the value of k, was found to be a small percentage of the total bandwidth consumption.

We observe a large spike at the beginning of each experiment, when we have just opened the browser and loaded the Twitter page. For instance, bandwidth consumption reaches 54 Kbps for $k = 100$, 29 Kbps for $k = 50$, and 7 Kbps

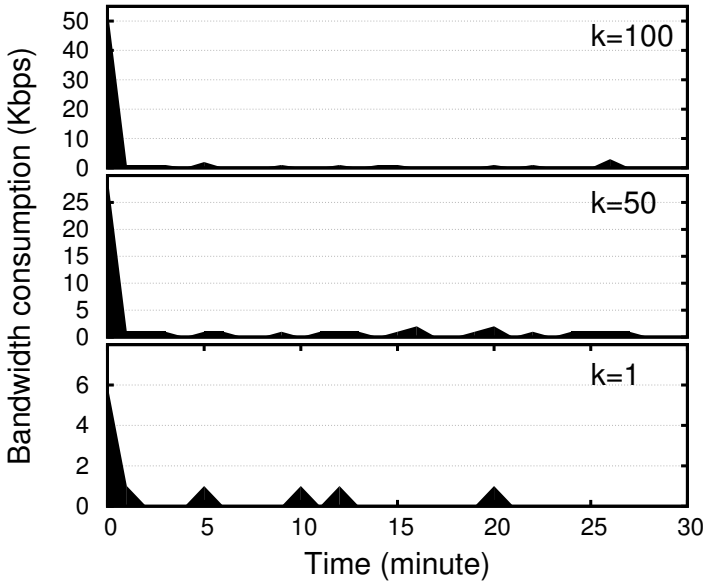
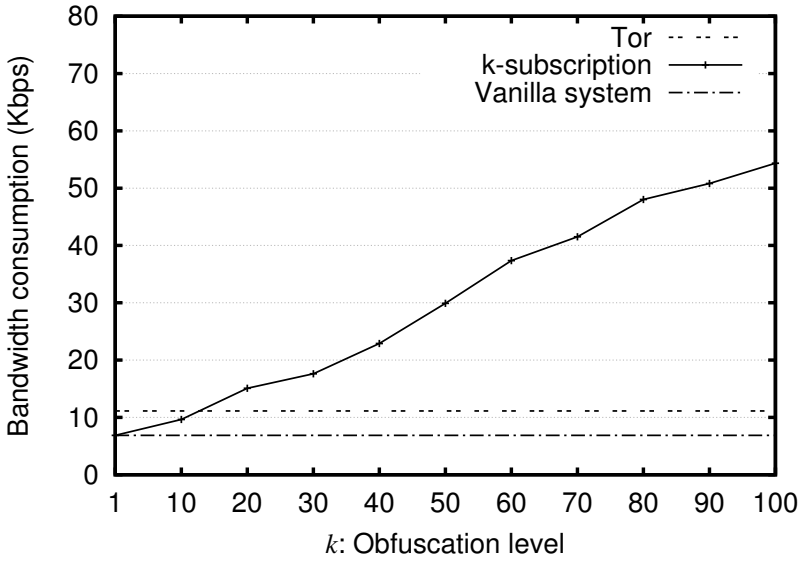


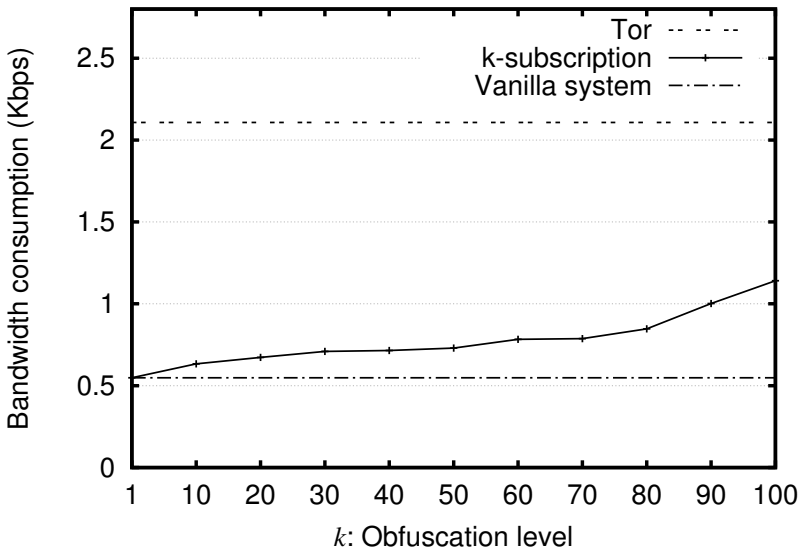
Figure 8.3: Bandwidth consumed for a user receiving tweets as a function of time.

for the vanilla system at the first second. During this initialization stage Twitter downloads all the necessary content (widgets, scripts, CSS, profile images, *etc.*). At this stage, *k-subscription* downloads lot of tweets from all k channels. As we discard most of the tweets (belonging to noise channels) and keep only tweets from channels that a user is interested in, we always download a larger chunk of tweets to be able to compound the user's actual timeline. For this reason we observe a relatively higher spike as k increases. To improve browsing latency, we transparently increase the page size to receive more tweets. Note that profile images are cacheable, so *k-subscription* downloads the additional images (depending on k) only once, without affecting the overall bandwidth consumption. After the initial spike in the first few seconds, we see constantly low bandwidth consumption, which correspond to the low rate of incoming tweets. The average consumption in this 30-minute interval is 1.14 Kbps for $k = 100$, 0.71 Kbps for $k = 50$, and 0.54 Kbps for $k = 1$. Overall, we see that the total bandwidth consumed by *k-subscription* is not really an issue even if the user follows as many as 100 channels.

To evaluate the effect of the obfuscation level on bandwidth consumption while browsing Twitter with *k-subscription*, we plot in Figure 8.4(a) and Figure 8.4(b) the bandwidth consumption as a function of k for two different stages: (i) when the user loads Twitter and downloads her timeline, which consists of the latest 20 tweets from the channels she is interested in, and (ii) when Twitter is idle and just receives new incoming tweets for 30 minutes.



(a) Initialization stage (load 20 tweets)



(b) Idle stage (download incoming tweets)

Figure 8.4: Bandwidth consumption with *k-subscription*, Tor and vanilla system.

We see that the overhead is very low, even for large k, and can be easily handled by a home DSL or even a mobile connection. The bandwidth consumption is much lower in the idle stage, as expected, due to the low number of tweets per second, as shown in Figure 8.2. The increased bandwidth during initialization is because *k-subscription* asks for more tweets to display the default page of 20 tweets only from channels that user is interested in. However, the initialization lasts for just few seconds, e.g., 7.7 seconds for $k = 100$ and 2.8 seconds for the vanilla system. Thus, the increased bandwidth in Figure 8.4(a) corresponds to short term spikes, while the low bandwidth in idle stage (see Figure 8.4(b)) corresponds to much longer periods, as the user keeps Twitter's page open in the browser.

In Figures 8.4(a) and 8.4(b) we also compare the bandwidth consumption of *k-subscription* with a Tor browser. Although Tor offers a completely different type of anonymity than *k-subscription*, it could be used together with a fake Twitter account as a different approach to hide user's interests. Thus, we evaluate *k-subscription* using the performance of Tor with a fake Twitter account as a baseline case. We see that Tor adds an additional bandwidth overhead due to its data encapsulation. In particular, the average packet size of Twitter traffic over Tor is 789 bytes, when the vanilla Twitter traffic has an average packet size of 239 bytes. This is the main reason why during the idle stage the bandwidth consumption of Tor is quite higher than the consumption of *k-subscription*, e.g., two times higher when $k = 90$. During the initialization stage, Tor has a higher bandwidth consumption than *k-subscription* with values of k up to 10, and lower consumption when k exceeds 10. This is due to the increased number of tweets downloaded at startup by *k-subscription* with high k values to construct a full page of useful tweets. However, as the initialization stage lasts only for few seconds, compared with idle stage, *k-subscription* adds less overhead in terms of bandwidth.

When *k-subscription* compounds the user's timeline, it continues to download tweets in the background until it reaches a certain number, which is constant for each k value. This way, *k-subscription* avoids leaking any information that Twitter could analyze to find out the channels a user is interested in.

8.5 Browsing Latency

In our next experiment we set out to explore the latency that *k-subscription* imposes to user's browsing experience. We instrumented our browser extension to measure the latency from the time that a user asks for one or more tweets till the time that the browser actually displays the relative information in the page, excluding any tweets from noise channels. This latency includes the time spent in network for downloading tweets, as well as the time spent in the CPU for excluding the noise and rendering the page. Note that the user's timeline is fully rendered when all the 20 tweets needed are received, despite the fact that more tweets are downloaded in the background to hide the actual user's interests.

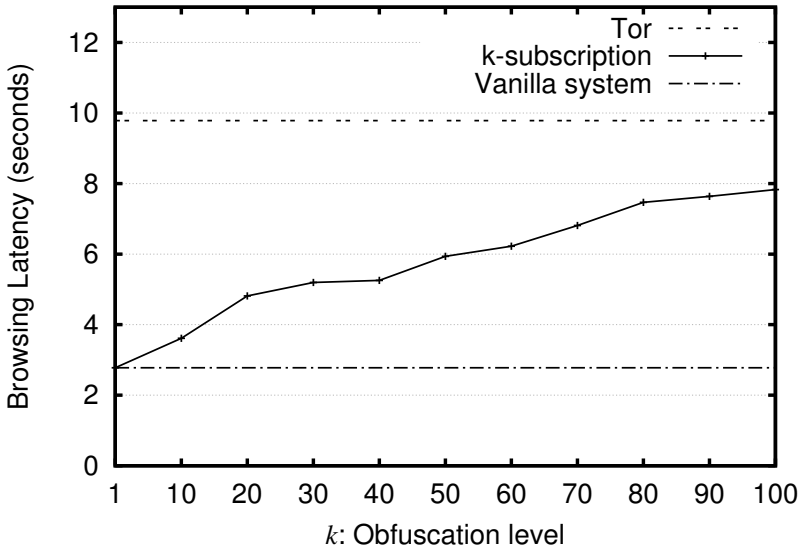


Figure 8.5: Browsing latency as a function of k when a user opens Twitter's main page. There is no noticeable latency for the incoming tweets when the browser remains open (idle case).

Figure 8.5 shows the latency for displaying a page with *k-subscription* for several values of k when the user opens Twitter and loads her timeline. We see that the latency for downloading and displaying a full page with 20 tweets slightly increases with the number of noise channels, reaching to 7.7 seconds for $k = 100$ when without *k-subscription* (see $k = 1$) the latency to display the same page is 2.8 seconds. Therefore, a slight delay of less than 5 seconds is not expected to significantly affect the user's browsing experience, while, at the same time, it enhances her privacy. Selecting a smaller number of noise channels results in even lower latency. Note that this small delay is only observed at the initialization stage, due to the increased number of tweets needed to construct the user's actual timeline. When the browser remains open (idle stage) we do not observe any noticeable delay to render the incoming tweets, even at very high values of k. If an incoming tweet belongs to a noise channel we just drop it, else it is immediately given to user with no further delay. Thus, our approach does not impose any significant overhead to the browsing latency. In Figure 8.5 we also compare the browsing latency of *k-subscription* and Tor. We see that Tor requires a much higher latency to display Twitter's page, close to 10 seconds. This is due to the longer path from user to Twitter through the anonymization network.

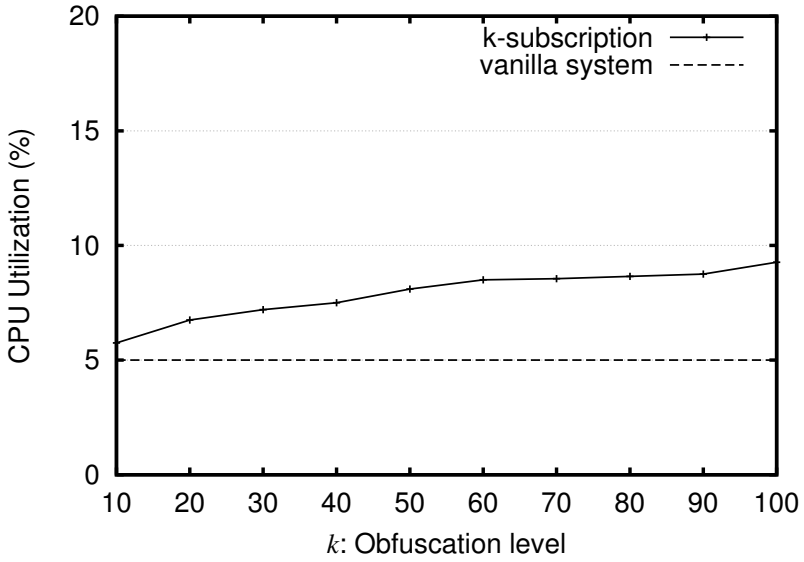


Figure 8.6: CPU usage for Initialization stage as a function of k.

During the previous experiments we measured the CPU load of the browser, using the Linux's `time` utility. As Figure 8.6 shows, the CPU load during the Initialization stage was negligible for all values of k, always less than 10%, even for $k = 100$, when it was always less than 3% during the Idle stage Thus, our *k-subscription* browser extension does not add any considerable CPU overhead to the system.

Chapter 9

Discussion

9.1 Anonymity

When users follow several "noise" channels to cover the channel(s) they are really interested in, their microblogging service profile might seem "strange" to friends and relatives. Indeed, a user following illness-related channels or bankruptcy-related channels may give the wrong impressions to friends and family who may start worrying about the health or financial stability of the particular user. In such cases, rumors do not take long to start and may have an adverse impact on the user. We view at least two ways to deal with this issue: (i) The service may provide privacy options, which may prohibit other people from finding which channels a particular user may follow. Some microblogging services like Twitter offer several options for managing the accounts that each user follows, e.g., by organizing them in several lists. This way, a separate list with channels available to the public, and another list with private channels available only to the user, could be possible. This would allow the user to hide her sensitive interests from other users of the social network – but it cannot hide them from the social networking service. (ii) The user may create a dummy account, which cannot be linked to her and/or her friends and relatives. Note that a separate dummy account cannot protect user A from the microblogging service. Indeed, the microblogging social networking service may identify A using several other tracking mechanisms, such as IP address tracking and cookies, even when A is logged on using the dummy account. This account will be used to follow channels with sensitive information.

9.2 What Is Effective Noise?

It is interesting to try to understand what kind of "noise" channels are effective in hiding a user's real interests. A naive approach would think that adding lots of noise channels would hide more effectively a user's real interests. However, it is not how many noise channels a user A adds in order to hide a sensitive channel C, but how many other users include C among their "noise" channels. Previous

approaches in search engine queries obfuscation encouraged users to add lots of "noise" queries, but made no effort to make sure that a query submitted by a particular user would also be submitted by other users as well. As a result, there were queries submitted by only one user. Such queries could clearly skyrocket the disclosure probability very close to 1. Indeed, if a query has been submitted by a very small number of users, it is not difficult for the search engine to find out which users are probably interested in this query.

To avoid this problem, our approach makes sure that several users will use each sensitive channel C as a "noise" channel. Furthermore, by including C in a set of sensitive channels, and by asking all participating users to select their "noise" channels from this set, we make sure that C will also be selected by several users.

9.3 Disappearing Channels

It is theoretically possible that some of the channels included in set S will disappear at some point in time. Indeed, people may choose to delete their accounts, corporations may go out of business, or organizations may change their focus. In such cases, their channel (say D) in the microblogging service will be deleted or it will become inactive. This will create two kinds of problems:

First, people who used to follow channel D cannot just stop following it. Moreover, they cannot stop following the noise channels, which were selected to be followed along with D as well. If the users stop following the above channels, the microblogging service will immediately correlate D's disappearance with the users' change of following patterns, and figure out that the users were interested in channel D. Fortunately, this not particularly worrisome. Indeed, the users should continue following D along with their selected noise channels. They will just pay a small overhead in downloading the channels they do not need anymore.

The second problem is that people who included D among their noise channels may be more exposed to microblogging service. Indeed, if their noise channels start disappearing one after the other, it seems that the microblogging service will be in a better position to find the exact channel they are really interested in. An obvious approach would be to add other noise channels in the place of the disappeared ones. Unfortunately, this is not correct, as the microblogging service will easily figure out that both the disappeared ones as well as the recently followed ones are noise channels. Surprisingly, the best approach for the users is to *do nothing*: they should just keep following the channels even if they are disappearing one after the other all the way to the last one. The key observation here is that both, the users who are interested in D and the users who are not interested in D but have just included D as noise, should do the same thing: *nothing*. In this way, the microblogging service will not be able to differentiate which users are interested in D and which are not.

Chapter 10

Conclusion

Microblogging services such as Twitter, Tumblr, or Fanfou, enable users to have timely access to their information needs through a publish-subscribe model. As users declare all the channels they are interested in following, the microblogging service is able to gather all their interests, including possible privacy-sensitive domains. (possibly among other things) personal preferences, political and religious beliefs, and maybe even medical history. Although this is a simple, useful, and timely model, it creates major privacy concerns.

To remedy this situation, we propose *k-subscription*: an obfuscation-based approach that encourages the users to follow $k - 1$ additional "noise" channels for each channel they are really interested in following. We present a detailed analysis of our approach and show that, by fine-tuning the k parameter we are able to reduce the confidence the microblogging service has in distinguishing the channels each user is actually interesting in. We have developed a prototype implementation as an extension for the Chrome browser using Twitter as a case study. Our experimental evaluation shows that users may easily follow hundreds of noise channels with minimal run-time overhead when they receive news they are interested in.

We believe that as an ever-increasing number of users turn to microblogging services for their daily and timely information needs, privacy concerns will continue to escalate, and solutions such as *k-subscription* will become increasingly more important.

Bibliography

[1] Sadia Afroz, Michael Brennan, and Rachel Greenstadt. Detecting Hoaxes, Frauds, and Deception in Writing Style Online. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, pages 461–475, Washington, DC, USA, 2012. IEEE Computer Society.

[2] Demetris Antoniades, Iasonas Polakis, Georgios Kontaxis, Elias Athanasopoulos, Sotiris Ioannidis, Evangelos P. Markatos, and Thomas Karagiannis. We.b: The web of short urls. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 715–724, New York, NY, USA, 2011. ACM.

[3] Dustin Bachrach, Christopher Nunu, Dan S. Wallach, and Matthew Wright. #h00t: Censorship resistant microblogging. *CoRR*, abs/1109.6874, 2011.

[4] Ero Balsa, Carmela Troncoso, and Claudia Diaz. Ob-pws: Obfuscation-based private web search. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 491–505, Washington, DC, USA, 2012. IEEE Computer Society.

[5] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web mixes: A system for anonymous and unobservable internet access. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, pages 115–129, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[6] Yazan Boshmaf, Ildar Muslukhov, Konstantin Beznosov, and Matei Ripeanu. Key challenges in defending against malicious socialbots. In *Proceedings of the 5th USENIX Conference on Large-Scale Exploits and Emergent Threats*, LEET'12, pages 12–12, Berkeley, CA, USA, 2012. USENIX Association.

[7] Elie Bursztein, Matthieu Martin, and John Mitchell. Text-based captcha strengths and weaknesses. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 125–138, New York, NY, USA, 2011. ACM.

[8] Zi Chu, Steven Gianvecchio, Haining Wang, and Sushil Jajodia. Who is tweeting on twitter: Human, bot, or cyborg? In *Proceedings of the 26th*

Annual Computer Security Applications Conference, ACSAC '10, pages 21–30, New York, NY, USA, 2010. ACM.

[9] Emiliano De Cristofaro, Claudio Soriente, Gene Tsudik, and Andrew Williams. Hummingbird: Privacy at the time of twitter. *IACR Cryptology ePrint Archive*, 2011:640, 2011.

[10] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[11] Domingo-Ferrer, Josep and Solanas, Agusti and Castellà-Roca, Jordi. h(k)-private information retrieval from privacy-uncooperative queryable databases. *Online Information Review*, 33(4):720–744, 2009.

[12] Peter Eckersley. How unique is your web browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, pages 1–18, Berlin, Heidelberg, 2010. Springer-Verlag.

[13] Y. Elovici, C. Glezer, and B. Shapira. Enhancing Customer Privacy While Searching for Products and Services on the World Wide Web. *Internet Research*, 15:378 – 399, 2005.

[14] Daniel C. Howe and Helen Nissenbaum. TrackMeNot: Resisting surveillance in web search. In Ian Kerr, Valerie Steeves, and Carole Lucock, editors, *Lessons from the Identity Trail: Anonymity, Privacy, and Identity in a Networked Society*, chapter 23, pages 417–436. Oxford University Press, Oxford, UK, 2009.

[15] Rosie Jones, Ravi Kumar, Bo Pang, and Andrew Tomkins. "i know what you did last summer": Query logs and user privacy. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 909–914, New York, NY, USA, 2007. ACM.

[16] H. Kido, Y. Yanagisawa, and T. Satoh. An anonymous communication technique using dummies for location-based services. In *Pervasive Services, 2005. ICPS '05. Proceedings. International Conference on*, pages 88–97. IEEE Computer Society, 2005.

[17] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. L-diversity: privacy beyond k-anonymity. In *Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on*, pages 24–24, New York, NY, USA, 2006. ACM.

[18] Jonathan R. Mayer and John C. Mitchell. Third-party web tracking: Policy and technology. In *Proceedings of the 2012 IEEE Symposium on Security and*

Privacy, SP '12, pages 413–427, Washington, DC, USA, 2012. IEEE Computer Society.

[19] Greg Mori and Jitendra Malik. Recognizing objects in adversarial clutter: Breaking a visual captcha. In *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'03, pages 134–141, Washington, DC, USA, 2003. IEEE Computer Society.

[20] S. Mori, H. Nishida, and H. Yamada. *Optical Character Recognition*. John Wiley & Sons, Inc., 1999.

[21] Mummoorthy Murugesan and Chris Clifton. Providing privacy through plausibly deniable search. In *SDM*, pages 768–779. SIAM, 2009.

[22] Arvind Narayanan, Hristo Paskov, Neil Zhenqiang Gong, John Bethencourt, Emil Stefanov, Eui Chul Richard Shin, and Dawn Song. On the feasibility of internet-scale author identification. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 300–314, Washington, DC, USA, 2012. IEEE Computer Society.

[23] Hyoungmin Park and Kyuseok Shim. Approximate algorithms for k-anonymity. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 67–78, New York, NY, USA, 2007. ACM.

[24] Sai Teja Peddinti and Nitesh Saxena. On the effectiveness of anonymizing networks for web search privacy. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, pages 483–489, New York, NY, USA, 2011. ACM.

[25] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, November 1998.

[26] Ricardo Stuven. A browser extension to filter tweets, 2012. `https://github.com/rstuven/OpenTweetFilter`.

[27] RT. Privacy betrayed: Twitter sells multi-billion tweet archive. `http://rt.com/news/twitter-sells-tweet-archive-529/`, 2012.

[28] Pierangela Samarati and Latanya Sweeney. Generalizing data to provide anonymity when disclosing information (abstract). In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, pages 188–, New York, NY, USA, 1998. ACM.

[29] P.Y. Simard. Using Machine Learning to Break Visual Human Interaction Proofs (HIPs). *Advances in Neural Information Processing Systems*, 2004.

[30] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Detecting spammers on social networks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 1–9, New York, NY, USA, 2010. ACM.

[31] Latanya Sweeney. K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, October 2002.

[32] J.W. Turner. Countermeasure Radar Chaff, 1970. US Patent 3,544,997.

[33] J. Yan and A.S. El Ahmad. Breaking visual captchas with naive pattern recognition algorithms. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 279–291. IEEE Computer Society, 2007.

[34] Jeff Yan and Ahmad Salah El Ahmad. A low-cost attack on a microsoft captcha. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 543–554, New York, NY, USA, 2008. ACM.

[35] Chao Yang, Robert Harkreader, Jialong Zhang, Seungwon Shin, and Guofei Gu. Analyzing spammers' social networks for fun and profit: A case study of cyber criminal ecosystem on twitter. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 71–80, New York, NY, USA, 2012. ACM.

[36] Shaozhi Ye, Felix Wu, Raju Pandey, and Hao Chen. Noise injection for search privacy protection. In *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 03*, CSE '09, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.