Tweeting with Hummingbird: Privacy in Large-Scale Micro-Blogging OSNs

Emiliano De Cristofaro
PARC
edc@parc.com

Claudio Soriente
ETH Zurich
claudio.soriente@inf.ethz.ch

Gene Tsudik
UC Irvine
gene.tsudik@uci.edu

Andrew Williams
UC Irvine
andrewmw@uci.edu

Abstract

In recent years, micro-blogging Online Social Networks (OSNs), such as Twitter, have taken the world by storm, now boasting over 100 million subscribers. As an unparalleled stage for an enormous audience, they offer fast and reliable diffusion of pithy tweets to great multitudes of information-hungry and always-connected followers with short attention spans. At the same time, this appealing information gathering and dissemination paradigm prompts some important privacy concerns about relationships between tweeters, followers and interests of the latter.

In this paper, we assess privacy in today's Twitter-like OSNs and describe an architecture and a trial implementation of a privacy-preserving service called Hummingbird – a variant of Twitter that protects tweet contents, hashtags and follower interests from the (potentially) prying eyes of the central server. We argue that, although inherently limited by Twitter's mission of scalable information dissemination, the attainable degree of privacy is valuable. We demonstrate, via a working prototype, that Hummingbird's additional costs are tolerably low. We also sketch out some viable enhancements that might offer even better privacy in the long term.

1 Introduction

Online Social Networks (OSNs) offer multitudes of people a means to communicate, share interests, and update others about their current activities. Alas, as their proliferation increases, so do privacy concerns with regard to the amount and sensitivity of disseminated information. Popular OSNs (such as Facebook, Twitter and Google+) provide customizable "privacy settings", i.e., each user can specify other users (or groups) that can access her content. Information is often classified by categories, e.g., personal, text post, photo or video. For each category, the account owner can define a coarse-grained access control list (ACL). This strategy relies on the trustworthiness of OSN providers and on users appropriately controlling access to their data. Therefore, users need to trust the service not only to respect their ACLs, but also to store and manage all accumulated content.

OSN providers are generally incentivized to safeguard users' content, since doing otherwise might tarnish their reputation and/or result in legal actions. However, user agreements often include clauses that let providers mine user content, e.g., deliver targeted advertising [11] or re-sell information to third-party services. Moreover, privacy risks are exacerbated by the common practice of caching content and storing it off-line (e.g., on tape

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

backups), even after users explicitly delete it. Thus, the threat to user privacy becomes permanent. Therefore, it appears that a more effective (or at least an alternative) way of addressing privacy in OSNs is by delegating control over content to its owners, i.e., the end-users. Towards this goal, the security research community has already proposed several approaches [2, 8, 12] that allow users to explicitly authorize "friends" to access their data, while hiding content from the provider and other entities.

However, the meaning of *relationship*, or *affinity*, among users differs across OSNs. In some, it is not based on any real-life trust. For example, micro-blogging OSNs, such as Twitter and Tumblr, are based on (short) information exchanges among users who might have no common history, no mutual friends and possibly do not trust each other. In such settings, a user publishes content labeled with some "tags" that help others search and retrieve content of interest. Furthermore, privacy in micro-blogging OSNs is not limited to content. It also applies to potentially sensitive information that users (subscribers or followers) disclose by searches and interests. Specifically, tags used to label and retrieve content might leak personal habits, political views, or even health conditions. This is particularly worrisome considering that authorities are increasingly monitoring and subpoenaing social network content [6]. Therefore, we believe that privacy mechanisms for micro-blogging OSNs, such as Twitter, should be designed differently from personal affinity-based OSNs, such as Facebook.

1.1 Motivation

Twitter is clearly the most popular micro-blogging OSN today. It lets users share short messages (tweets) with their "followers" and enables enhanced content search based on keywords (referred to as *hashtags*) embedded in tweets. Over time, Twitter has become more than just a popular micro-blogging service. Its pervasiveness makes it a perfect means of reaching large numbers of people through their always-on mobile devices. Twitter is also the primary source of information for untold millions (individuals as well as various entities) who obtain their news, favorite blog posts or security announcements via simple 140-character tweets.

Users implicitly trust Twitter to store and manage their content, including: tweets, searches, and interests. Thus, Twitter possesses complex and valuable information, such as tweeter-follower relationships and hashtag frequencies. As mentioned above, this prompts privacy concerns. User interests and trends expressed by the "Follow" button represent sensitive information. For example, looking for tweets with a hashtag #TeaParty, (rather than, say, #BeerParty), might expose one's political views. A search for #HIVcure might reveal one's medical condition and could be correlated with the same user's other activity, e.g., repeated appearances (obtained from a geolocation service, such as Google Latitude) of the user's smartphone next to a clinic. Based on its enormous popularity, Twitter has clearly succeeded in its main goal of providing a ubiquitous real-time push-based information sharing platform. However, we believe that it is time to re-examine whether it is reasonable to trust Twitter to store and manage content (tweets) or search criteria, as well as to enforce user-defined ACLs.

1.2 Overview

This paper describes Hummingbird: a privacy-enhanced variant of Twitter. Hummingbird retains key features of Twitter while adding several privacy-sensitive ingredients. Its goal is two-fold:

1. Private fine-grained authorization of followers: a tweeter encrypts a tweet and chooses who can access it, e.g., by defining an ACL based on tweet content.

2. Privacy for followers: they subscribe to arbitrary hashtags without leaking their interests to any entity. That is, Alice can follow all #OccupyWS tweets from the New York Times (NYT) such that neither Twitter nor NYT learns her interests.

Hummingbird can be viewed as a system composed of several cryptographic protocols that allow users to tweet and follow others' tweets with privacy. We acknowledge, from the outset, that privacy features advocated in

this paper would affect today's business model of a micro-blogging OSN. Since, in Hummingbird, the provider does not learn tweet contents, current revenue strategies (e.g., targeted advertising) would be difficult to realize. Consequently, it would be both useful and interesting to explore economic incentives for providing privacy-friendly services and not just in the context of micro-blogging OSNs. However, this topic is beyond the scope of this paper.

To demonstrate Hummingbird's practicality, we implemented it as a web site on the server side. On the user side, we implemented a Firefox extension to access the server, by making cryptographic operations transparent to the user. Hummingbird imposes minimal overhead on users and virtually no extra overhead on the server; the latter simply matches tweets to corresponding followers.

2 Privacy in Twitter

This section overviews Twitter, discusses fundamental privacy limitations and defines privacy in micro-blogging OSNs.

2.1 Twitter

As the most popular micro-blogging OSN, Twitter (http://www.twitter.com) boasts over 100 million active users worldwide, including: journalists, artists, actors, politicians, socialites, regular folks and crackpots of all types, as well as government agencies, NGOs and commercial entities [14]. Its users communicate via 140-character messages, called *tweets*, using a simple web interface. Posting messages is called *tweeting*. Users may subscribe to other users' tweets; this practice is known as *following*. Basic Twitter terminology is as follows:

- A user who posts a tweet is a *tweeter*.
- A user who follows others' tweets is a *follower*.
- The centralized entity that maintains profiles and matches tweets to followers is simply *Twitter*.

Tweets are labeled and retrieved (searched) using ***hashtags***, i.e., strings prefixed by a "#" sign. For example, a tweet: "I don't care about #privacy on #Twitter" would match any search for hashtags "#privacy" or "#Twitter". An "@" followed by a user-name is utilized for mentioning, or replying to, other users. Finally, a tweet can be re-published by other users, and shared with one's own followers, via the so-called *re-tweet* feature. Tweets are *public* by default: any registered user can see (and search) others' public tweets. These are also indexed by third party services – such as Google – and can be accessed by application developers through a dedicated streaming API. All public tweets are also posted on a public website (http://twitter.com/public_timeline), that keeps the tweeting "timeline" and shows twenty most recent messages. Tweeters can restrict availability of their tweets by making them "private" – accessible only to authorized followers [1]. Tweeters can also revoke such authorizations, using (and trusting) Twitter's *block* feature. Nonetheless, whether a tweet is public or private, Twitter collects all of them and forwards them to intended recipients. Thus, Twitter has access to all information within the system, including: tweets, hashtags, searches, and relationships between tweeters and their followers. Although this practice facilitates dissemination, availability, and mining of tweets, it also intensifies privacy concerns stemming from exposure of information.

Defining privacy in a Twitter-like system is a challenging task. Our definition revolves around the server (i.e., Twitter itself) that needs to match tweets to followers while learning as little as possible about both. This would be trivial if tweeters and followers shared secrets [15]. It becomes more difficult when they have no common secrets and do not trust each other.

2.2 Built-in Limitations

From the outset, we acknowledge that privacy attainable in Twitter-like systems is far from perfect. Ideal privacy in micro-blogging OSNs can be achieved only if no central server exists: all followers would receive all tweets

and decide, in real-time, which are of interest. Clearly, this would be unscalable and impractical in many respects. Thus, a third-party server is needed. The main reason for its existence is the matching function: it binds incoming tweets to subscriptions and forwards them to corresponding followers. Although we want the server to learn no more information than an adversary observing a secure channel, the very same matching function precludes it. Similarly, the server learns whenever multiple subscriptions match the same hashtag in a tweet. Preventing this is not practical, considering that a a tweeter's single hashtag might have a very large number of followers. It appears that the only way to conceal the fact that multiple followers are interested in the same hashtag (and tweet) is for the tweeter to generate a distinct encrypted (*tweet, hashtag*) pair for each follower. This would result in a linear expansion (in the number of followers of a hashtag) for each tweet. Also, considering that all such pairs would have to be uploaded at roughly the same time, even this unscalable approach would still let the server learn that – with high probability – the same tweet is of interest to a particular set of followers.

Note that the above is distinct from server's ability to learn whether a given subscription matches the same hashtag in multiple tweets. As we discuss in [4], the server can be precluded from learning this information, while incurring a bit of extra overhead. However, it remains somewhat unclear whether the privacy gain would be worthwhile.

2.3 Privacy Goals and Security Assumptions

Our privacy goals are commensurate with aforementioned limitations.

- Server: learns minimal information beyond that obtained from performing the matching function. We allow it to learn which, and how many, subscriptions match a hashtag; even if the hashtag is cryptographically transformed. Also, it learns whether two subscriptions for the same tweeter refer to the same hashtag. Furthermore, it learns whenever two tweets by the same tweeter carry the same hashtag.
- Tweeter: learns who subscribes to its hashtags but not *which* hashtags have been subscribed to.
- Follower: learns nothing beyond its own subscriptions, i.e., it learns no information about any other subscribers or any tweets that do not match its subscriptions.

Our privacy goals, coupled with the desired features of a Twitter-like system, prompt an important assumption that the server must adhere to the **Honest-but-Curious (HbC)** adversarial model. Specifically, although the server is assumed to faithfully follow all protocol specifications, it might attempt to passively violate our privacy goals. According to our interpretation, the HbC model precludes the server from creating "phantom" users. In other words, the server does not create spurious (or fake) accounts in order to obtain subscriptions and test whether they match other followers' interests.

The justification for this assertion is as follows. Suppose that the server creates a phantom user for the purpose of violating privacy of genuine followers. The act of creation itself does not violate the HbC model. However, when a phantom user contacts s a genuine tweeter in order to obtain a subscription, a protocol transcript results. This transcript testifies to the existence of a spurious user (since the tweeter can keep a copy) and can be later used to demonstrate server misbehavior.

We view this assumption as unavoidable in any Twitter-like OSN. The server provides the most central and the most valuable service to large numbers of users. It thus has a valuable reputation to maintain and any evidence, or even suspicion, of active misbehavior (i.e., anything beyond HbC conduct) would result in a significant loss of trust and a mass exodus of users.

2.4 Definitions

We now provide some definitions to capture privacy loss that is unavoidable in Hummingbird in order to efficiently match tweets to subscriptions. Within a tweet, we distinguish between the actual message and its hashtags, i.e., keywords used to tag and identify messages.

Tweeter Privacy. An encrypted tweet that includes a hashtag ht should leak no information to any party that has not been authorized by the tweeter to follow it on ht. In other words, only those authorized to follow the tweeter on a given hashtag can decrypt the associated message. For its part, the server learns whenever multiple tweets from a given tweeter contain the same hashtag.

Follower Privacy. A request to follow a tweeter on hashtag ht should disclose no information about the hashtag to any party other than the follower. That is, a follower can subscribe to hashtags such that tweeters, the server or any other party learns nothing about follower interests. However, the server learns whenever multiple followers are subscribed to the same hashtag of a given tweeter.

Matching Privacy. The server learns nothing about the content of matched tweets.

3 Private Tweeting in Hummingbird

In this section, we present Hummingbird architecture and protocols.

3.1 Architecture

Hummingbird architecture mirrors Twitter's, involving one central server and an arbitrary number of registered users, that publish and retrieve short text-based messages. Publication and retrieval is based on a set of hashtags that are appended to the message or specified in the search criteria.

Similar to Twitter, Hummingbird involves three types of entities:

1. ***Tweeters*** post messages, each accompanied by a set of hashtags that are used by others to search for those messages. For example, Bob posts a message: "I care about #privacy" where "#privacy" is the associated hashtag.
2. ***Followers*** issue "follow requests" to any tweeter for any hashtag of interest, and, if a request is approved, receive all tweets that match their interest. For instance, Alice who wants to follow Bob's tweets with hashtag "#privacy" would receive the tweet: "I care about #privacy" and all other Bob's tweets that contain the same hashtag.
3. ***Hummingbird Server*** (HS) handles user registration and operates the Hummingbird web site. It is responsible for matching tweets with follow requests and delivering tweets of interest to users.

3.2 Design Overview

Unlike Twitter, access to tweets in Hummingbird is restricted to authorized followers, i.e., they are hidden from HS and all non-followers. Also, all follow requests are subject to approval. Whereas, in Twitter, users can decide to approve all requests automatically. Also, Hummingbird introduces the concept of ***follow-by-topic***, i.e., followers decide to follow tweeters and specify hashtags of interest. This feature is particularly geared for following high-volume tweeters, as it filters out "background noise" and avoids inundating users with large quantities of unwanted content. For example, a user might decide to follow the New York Times on #politics, thus, not receiving NYT's tweets on, e.g., #cooking, #gossip, etc. Furthermore, follow-by-topic might allow tweeters to charge followers a subscription fee, in order to access premium content. For example, Financial Times could post tweets about stock market trends with hashtag #stockMarket and only authorized followers who pay a subscription fee would receive them.

Key design elements are as follows:

1. Tweeters encrypt their tweets and hashtags.
2. Followers can *privately* follow tweeters on one or more hashtags.
3. HS can *obliviously* match tweets to follow requests.
4. Only authorized (previously subscribed) followers can decrypt tweets of interest.

At the same time, we need to minimize overhead at HS. Ideally, privacy-preserving matching should be as fast and as scalable as its non-private counterpart.

Intuition. At the core of Hummingbird architecture is a simple Oblivious PRF (OPRF) technique. Informally, an OPRF [7, 3, 9, 5, 10] is a two-party protocol between sender and receiver. It securely computes $f_s(x)$ based on secret index s contributed by sender and input x – by receiver, such that the former learns nothing from the interaction, and the latter only learns $f_s(x)$.

Suppose Bob wants to tweet a message M with a hashtag ht. The idea is to derive an encryption key for a semantically secure cipher (e.g., AES) from $f_s(ht)$ and use it to encrypt M. (Recall that s is Bob's secret.) That is, Bob computes $k = H_1(f_s(ht))$, encrypts $Enc_k(M)$ and sends it to HS. Here, $H_1 : \{0,1\}^* \rightarrow \{0,1\}^{\tau_1}$ is a cryptographic hash function modeled as a random oracle and τ_1 is a polynomial function of the security parameter τ.

To follow Bob's tweets with ht, another user (Alice) must first engage Bob in an OPRF protocol where she plays the role of the receiver, on input ht, and Bob is the sender. As a result, Alice obtains $f_s(ht)$ and derives k that allows her to decrypt all Bob's tweets containing ht. Based on OPRF security properties, besides guaranteeing tweets' confidentiality, this protocol also prevents Bob from learning Alice's interests, i.e., he only learns that Alice is among his followers but not which hashtags are of interest to her. Alice and Bob do not run the OPRF protocol directly or in real time. Instead, they use HS as a conduit for OPRF protocol messages.

Once Alice establishes a follower relationship with Bob, HS must also efficiently and *obliviously* match Bob's tweets to Alice's interests. For this reason, we need a secure tweet labeling mechanism.

To label a tweet, Bob uses a PRF, on input of an arbitrary hashtag ht, to compute a cryptographic token t, i.e., $t = H_2(f_s(ht))$ where H_2 is another cryptographic hash function, modeled as a random oracle: $H_2 : \{0,1\}^* \rightarrow \{0,1\}^{\tau_2}$, with τ_2 polynomial function of the security parameter τ. This token is communicated to HS along with the encrypted tweet.

As discussed above, Alice must obtain $f_s(ht)$ beforehand, as a result of running an OPRF protocol with Bob. She then computes the same token t, and uploads it to HS. OPRF guarantees that t reveals no information about the corresponding hashtag. HS only learns that Alice is one of Bob's followers. From this point on, HS obviously matches Bob's tweets to Alice's interests. Upon receiving an encrypted tweet and an accompanying token from Bob, HS searches for the latter among all tokens previously deposited by Bob's followers. As a result, HS only learns that a tweet by Bob matches a follow request by Alice.

OPRF choice. Although Hummingbird does not restrict the underlying OPRF instantiation, we selected the construct based on Blind-RSA signatures (in ROM) from [5], since it offers the lowest computation and communication complexities. One side-benefit of using this particular OPRF is that it allows us to use standard RSA public key certificates. At the same time, the Hummingbird architecture can be seamlessly instantiated with any other OPRF construction.

Support for Multiple Hashtags. To ease presentation, we assumed that all tweets and follow requests contain a single hashtag, however, there is an easy extension for tweeting or issuing follow requests on multiple hashtags.

Suppose Bob wants to tweet a message M and associate it with n hashtags, $ht_1^*, \ldots, ht_n^*$: anyone with a follow request accepted on *any* of these hashtags should be able to read the message. We modify the tweeting protocol as follows: Bob selects $k^* \leftarrow_R \{0,1\}^{\tau_1}$, and computes $ct^* = Enc_{k^*}(M)$. He then computes: $\{\delta_i^* = H(ht_i^*)^{d_b} \bmod N_b\}_{i=1}^n$. Finally, Bob sends to HS $(ct^*, \{Enc_{H_1(\delta_i^*)}(k)\}_{i=1}^n, \{H_2(\delta_i^*)\}_{i=1}^n)$. The rest of the protocol involving matching at HS, as well as Alice's decryption, is straightforward, thus, we omit it.

Further, suppose Alice would like to follow Bob on *any* hashtags: $(ht_1, \ldots, ht_l)$: we only need to extend the OPRF interaction between them to l parallel executions and let Alice deposit the results to HS – the rest of the protocol is unmodified and is omitted to ease presentation.

4 System Prototype

We implemented Hummingbird as a fully functioning research prototype. It is available at http://sprout.ics.uci.edu/hummingbird. In this section, we demonstrate that: (1) by using efficient cryptographic mechanisms, Hummingbird offers a privacy-preserving Twitter-like messaging service, (2) Hummingbird introduces no overhead on the central service (HS) (thus raising no scalability concerns), and (3) Hummingbird's performance makes it suitable for real-world deployment.

4.1 Server-side

In the description below, we distinguish between server- and client-side components. Hummingbird's server-side component corresponds to HS, introduced in Section 3. It consists of three parts: (1) database, (2) JSP classes, and (3) Java back-end. We describe them below.

Database. Hummingbird employs a central database to store and access user accounts, encrypted tweets, follow requests, and profiles.

JSP Front-end. The visual component is realized through JSP pages. that allow users to seamlessly interact with a back-end engine via the web browser. Main web functionalities include: registration, login, issuing/accepting/finalizing a request to follow, tweeting, reading streaming tweets, and accessing user profiles.

Java Back-end. Hummingbird functionality is realized by a Java back-end running on HS. The back-end is deployed in Apache Tomcat. The software includes many modules; we omit their descriptions. The back-end mainly handles access to the database, populates web pages, and performs efficient matching of tweets to followers using off-the-shelf database mechanisms.

4.2 Client-side

Users interface with the system via the Hummingbird web site. We implemented each operation in Hummingbird as a web transaction. Users perform them from their own web browsers. However, several client-side cryptographic operations need to be performed outside the browser: to the best of our knowledge, there is no browser support for complex public-key operations such as those needed in OPRF computation.

To this end, we introduce, on the client-side, a small Java back-end to perform cryptographic operations. Then, we design a Firefox extension (HFE) to store users' keys and to *automatically* invoke appropriate Java code for each corresponding action. Its latest version is compatible with Firefox 3.x.x and is available from http://sprout.ics.uci.edu/hummingbird.

Client-side Java Back-end (CJB). Hummingbird users are responsible for generating their RSA keys, encrypting/decrypting tweets according to the technique presented in Section 3, and performing OPRF computations during follow request/approval. These cryptographic operations are implemented by a small Java back-end, CJB, included in the HFE presented below. CJB relies on the Java Bouncy Castle Crypto library.

Hummingbird Firefox Extension (HFE). As mentioned above, HFE is the interface between the web browser and the client-side Java back-end, included as part of the extension package. Extension code connects to it using Java LiveConnect [13]. Once installed, HFE is completely transparent to the user. HFE is used for:

Key management. At user registration, HFE automatically invokes RSA key generation code from CJB, stores (and optionally password-protects) public/private key in the extension folder, and lets browser report the public key to HS.

Following. For each of the three steps involved in requesting to follow a tweeter, the user is guided by Hummingbird web site, however, CJB code is executed to realize the corresponding cryptographic operations. This

is done automatically by HFE.

Tweet. When a user tweets, HFE transparently intercepts the message with its hashtags and invokes CJB code to encrypt the message and generate appropriate cryptographic tokens.

Read. Followers receive tweets from HS that match their interests, These tweets are encrypted (recall that matching is performed obliviously at HS). HFE automatically decrypts them using CJB code and replaces web page content with the corresponding cleartext.

5 Conclusion

This paper presented one of the first efforts to assess and mitigate erosion of privacy in modern micro-blogging OSNs. We analyzed privacy issues in Twitter and designed an architecture (called Hummingbird) that offers Twitter-like service with increased privacy guarantees for tweeters and followers alike. While the degree of privacy attained is far from perfect, it is still valuable considering current total lack of privacy and some fundamental limitations inherent to the large-scale centralized gather/scatter message dissemination paradigm. We implemented Hummingbird architecture and evaluated its performance. Since almost all cryptographic operations are conducted off-line, and none is involved to match tweets to followers, resulting costs and overhead are very low. Our work clearly does not end here. In particular, several extensions, including revocation of followers, anonymity for tweeters as well as unlinking same-hashtag tweets, require further consideration and analysis.

References

[1] Twitter Privacy Policy. https://twitter.com/privacy, 2011.

[2] F. Beato, M. Kohlweiss, and K. Wouters. Scramble! your social network data. In *PETS*, 2011.

[3] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable Proofs and Delegatable Anonymous Credentials. In *CRYPTO*, 2009.

[4] E. De Cristofaro, C. Soriente, G. Tsudik, and A. Williams. Hummingbird: Privacy at the Time of Twitter. In *S&P*, 2012.

[5] E. De Cristofaro and G. Tsudik. Practical Private Set Intersection Protocols with Linear Computational and Bandwidth Complexity. In *FC*, 2010. http://eprint.iacr.org/2009/491.

[6] K. Dozier. CIA Tracks Revolt by Tweet, Facebook. http://abcn.ws/uFdpVQ, 2011.

[7] M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword Search and Oblivious Pseudorandom Functions. In *TCC*, 2005.

[8] S. Guha, K. Tang, and P. Francis. NOYB: Privacy in Online Social Networks. In *WONS*, pages 49–54, 2008.

[9] S. Jarecki and X. Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *TCC*, 2009.

[10] S. Jarecki and X. Liu. Fast Secure Computation of Set Intersection. In *SCN*, 2010.

[11] H. Jones and J. Soltren. Facebook: Threats to privacy. In *Ethics and the Law on the Electronic Frontier Course*, 2005.

[12] W. Luo, Q. Xie, and U. Hengartner. FaceCloak: An Architecture for User Privacy on Social Networking Sites. In *CSE*, 2009.

[13] Mozilla Developer Network. LiveConnect. https://developer.mozilla.org/en/LiveConnect, 2011.

[14] Official Twitter Blog. One hundred million voices. http://blog.twitter.com/2011/09/one-hundred-million-voices.html, 2011.

[15] D. X. Song, D. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. In *S&P*, 2000.