Twistor: Anonymous Message Transfer through Twitter

by

Marjan Sadat Alavi

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

January 2016

Abstract

Anonymous communications is a long sought goal of dissidents, privacy advocates, and a host of other user communities. While a large number of systems have been proposed, those systems generally require large-scale communication infrastructure to be built in order to achieve a non-trivial amount of anonymity. However, the very nature of anonymity has meant that a business rationale for building such infrastructure is lacking.

Motivated to design a practical receiver anonymity network with large anonymity set sizes, we present Twistor: a new system for receiver-anonymous communications which leverages the Twitter social graph as the underlying anonymizing layer. To send a twist, a Twistor client first checks for reachability of its intended recipient, using local graph information maintained by interactions with the Twistor server. It then encrypts the message under the recipient's public key and posts the ciphertext to the corresponding user's timeline. Larger ciphertexts are encoded into an image, so as to conform with Twitter's 140 UTF-8 character limit.

Twistor clients are listening for Twistor posts and decrypt and repost when those they follow publish a new twist, except for when a TTL indicator is 0. Given self-reducibility properties of ElGamal, even if the adversary, e.g., Twitter, monitors all Twistor posts and re-posts that cascade from an initial post, the anonymity of the receiver and the confidentiality of the plaintext are reducible to the hardness of Decisional Diffie-Hellman problem.

Twistor derives its increase in size of the receiver anonymity set from asymmetric social connections combined with the publish-subscribe communication model in Twitter. Our aim is to achieve receiver anonymity set sizes on the order of hundreds of thousands.

In this thesis we describe our built system, the cost to the underlying infrastructure and the tradeoffs between those costs and the size of the anonymity set.

Preface

- Chapter 3 of this work is a product of weekly meetings with my supervisor Prof. Bill Aiello.
  Jean-Sébastien Légaré has also made some insightful suggestions during our system design; He also suggested an architecture for the interconnection between the user and the web server on Twistor client side.

- The implementation in Chapter 4 is done by myself, with some codes in Section 4.2 being written by Mira Leung. Some parts of those codes does not appear the the final version of our work; due to the change in the design.

- The evaluation in Chapter 5 is done by myself. In the preliminary version of this work, the cumulative distribution graphs for the number of per level Twitter followers were made by Maria Fung.
  Robert Reiss has also provided useful insights into Chapter 5.

Table of Contents

List of Figures

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Bill Aiello, for his patience and continuous generosity and support throughout my studies and research. Bill's expertise and understanding added considerably to my graduate experience at UBC. With his words of wisdom, advice, encouragement and positive perspective, he showed me how to achieve a true balance between professionalism and reality.

I am indebted to my fellows and friends Jean-Sébastien Légaré and Robert Reiss whom have offered inspiring and helpful comments on our weekly meetings and on dozens of my practice presentations.

I want to thank the undergraduate students I worked with: Maria Fung and Mira Leung for studying Twitter social graph and working on Twistor graph server code in the preliminary version of our research.

I am thankful to all faculties, fellows and friends at NSS lab: Prof. Ivan Beschastnikh, Nodir Kodirov, David Williams-King and all other NSS members, for making my graduate experience the one that I will cherish forever.

I thank Prof. Norm Hutchinson for spending his valuable time on reading my thesis and providing valuable feedback on my thesis. Also, I appreciate Prof. Andrew Warfield's wise and useful comments during my thesis presentation.

I truly appreciate my family's confidence in me. Through their genuine love, unconditional care and patience they helped me overcome setbacks and stay focused on my studies.

I am grateful to many other people at the Computer Science department, for their various forms of support during my graduate studies. Finally, I want to express my appreciation to the Faculty of Graduate and Postdoctoral Studies and University of British Columbia for their international tuition award and for providing me with all sort of amenities throughout my studies.

Dedication

"... Keep walking, though there is no place to get to.
Don't try to see through the distances. That's not for human beings.
Move within, but don't move the way fear makes you move.
Today, like every other day, we wake up empty and frightened.
Don't open the door to the study and begin reading.
Take down a musical instrument.
Let the beauty we love be what we do..."

Mawlana Rumi (1207-1273)

To my precious parents and siblings.

To my kind supervisor, Professor Bill Aiello.

Chapter 1

Introduction

Although data encryption is necessary for hiding the content of a message, just by observing the communication, the adversary can learn the identity of the communication endpoints. As a result, there has been plenty of research on designing systems which provide anonymity: [20], [21], [22], [25], [16], [26], [24], [23], [11], [12], [13], [14], [16] to name a few.

With the help of anonymity systems, activists, dissidents and investigative journalists living under oppressive regimes can communicate information with particular sources without being concerned about any social or governmental consequences.

Anonymity of a subject means the subject is not identifiable within a set of subjects, *the anonymity set* [1]. Anonymity systems can protect the identity of either one or both of the communication endpoints. Using sender anonymous systems, one can hide the identity of the original sender of the message, whereas receiver anonymity focuses on protecting the identity of the data recipient.

Depending on the strength of the design and using various traffic analysis techniques, the adversary might still be able to make some conclusions about the potential communication endpoints. In this respect, the group of people who are potential senders or receivers of a message, create the sender or receiver anonymity set respectively.

Various measures have been proposed in order to quantify the level of anonymity offered by anonymous communication systems. One of the most popular anonymity metrics is *Entropy*[1] [2], [3]. The entropy metric considers an adversary whom have assigned probabilistic information to different users as being the originators or recipients of a message.

Using this anonymity metric, if there be N users in the sender or receiver anonymity set and user i be the true sender or recipient of a message with probability p_i, then the entropy of the system $H(X)$ is calculated as follows:
$H(X) = -\sum_{i=1}^{N} p_i.log_2(p_i)$.

As such, the amount of information the adversary has learned through

[1]Based on Shannon's definition of Entropy.

her attack can be expressed as $log_2(N) - H(X)$. Clearly, the bigger the anonymity set, the smaller the decrease in entropy.

While a large number of anonymity systems have been proposed, the vast majority of those systems require large-scale communication infrastructure to be built in order to achieve a non-trivial amount of anonymity. However, most such systems have not reached the critical mass, because the very nature of anonymity has meant that a business rationale for building such infrastructure was lacking. TOR [20] pointed the way by showing that a practical, but not necessarily perfect, anonymity system could be built as an overlay on top of a general purpose, economically viable routing infrastructure that was already built–the global IP routing infrastructure. Nonetheless, some hesitate to use TOR as it is impossible to ascertain whether a sufficient number of TOR routers have been penetrated by an untrusted agent so as to de-anonymize some or all of a user's communication.

Motivated to build a feasible and practical anonymity system, in this thesis we present Twistor, a new system for achieving receiver anonymous communication. In this regard, we selected Twitter as a likely candidate to host our anonymous messaging system because of its reliability and its ability to forward messages to millions of people. Twitter is by far one of the most popular social networking service with over 300 million monthly active users as of August 2015 [19]. While most social networks are designed around the notion of mutual friendship with symmetric social connections, Twitter follows an asymmetric connection design. The asymmetry of social connections combined with Twitter's publish-subscribe communication model makes it very suitable for sending information to a very large audience through iterative reposts.

In summary, our contributions in this thesis are as follows:

- Leveraging the publish-subscribe communication model in asymmetric social networks to build a receiver anonymity system for social network users.

- Providing unlinkability of the communication between sender and receiver from the viewpoint of any online global adversary including the social network provider, even though sender and receiver know their corresponding communicating partner.

- Providing receiver anonymity set sizes on the order of hundreds of

thousands for low latency applications (with around 7 seconds end-to-end latency).

- Implementing the system on top of an existing asymmetric social networking infrastructure, i.e,Twitter.

Chapter 2 describes Twistor's threat model. Chapter 3 presents Twistor's system architecture. In Chapter 4, we describe the implementation details of our system. Chapter 5 presents the results of our experiments. Chapter 6 describes how Twistor relates to previous work. Chapter 7 concludes this thesis.

Chapter 2

Threat Model

2.1 Twitter Terminology

Twitter is one of the most popular asymmetric social networks where users exchange short messages, called Tweets, limited to 140 UTF-8 characters. Users who subscribe to a Twitter account, are called followers of that account which is their respective followee[2]. The real-time stream of Tweets shared by all followees of a user, is called the user's home timeline. When a Twitter user posts a tweet to her timeline, the tweet is displayed in the respective timelines of all her followers.

2.2 Threat Model

In anonymous communications, the adversary can either launch passive or active attacks. In Passive attacks, the adversary observes traffic passing through the relays in order to correlate inputs of a relay with its corresponding output. In active attacks, however, the adversary modifies some input messages while trying to trace the communication. In addition, the adversary can either be global or local. A global adversary has access to the whole communication system, while a local adversary can only control part of the resources.

We consider a global network adversary being able to monitor all tweets passed through the network, but whom can not subvert the cryptographic primitives. Also, we assume the Twitter graph is completely known to the adversary. The security property we want to maintain is that even if the adversary observes all the network communications, he should not be able to determine the identity of the intended recipient of a given message, within the receiver anonymity set known to him.

If some clients in the path misbehave by not processing and retweeting Twistor-related posts, this would cause DoS[3] only, but would not disrupt re-

[2]We use terms *followee* and *friend* interchangeably throughout this thesis.
[3]Denial of Service.

cipient anonymity. The same reasoning holds for adversary-controlled clients whom try to actively modify the messages passing through them.

For our anonymity purpose, we do not assume any trust on the provider of the social network other than it conforms to its own service level agreement. In other words, we assume Twitter provides correct information when queried for its graph information and that it does not misbehave when users publish some content to their subscribers.

Chapter 3

System Architecture

3.1 "Straw Man" Design

Consider a Twitter user whom is willing to send a message to her intended recipient, while protecting receiver's anonymity against any entity who is monitoring the communication. In a simple "straw man" design, the sender encrypts the message under the recipient's public key and posts the ciphertext into her timeline. All followers as well as any other entity whom might expect receiving a message from the user, could then try decrypting the message with their associated private keys. With a careful choice of the cryptosystem, the cipher texts would not reveal any information about the public key being used for encrypting the message[4]. This motivates potential recipients to try decrypting the ciphertext and at the same time doesn't leak any information about the intended recipient to the adversary.

Using this construction, the recipient anonymity set would consist of sender's intermediate followers as well as any Internet user in general whom visits sender's timeline. Note that Twitter access logs can easily reveal the IP addresses of non-followers or Internet users not connected to Twitter whom try to read sender's timeline, unless those users are connected through an anonymizer tool. Therefore, the effective recipient anonymity set for a typical Twitter user[5] would be rather small.

3.2 Twistor Design

If upon posting the ciphertext to her timeline, the user's followers act as relays by posting the ciphertext to their respective timelines, this would increase the receiver anonymity set size in a desirable way. This makes the basis for Twistor design. In this chapter, we describe the components of our improved design.

[4]This property is called "indistinguishability of the key" and not all cryptosystems maintain this property. ElGamal cryptosystem, which we have used in this work, satisfies this property [17].

[5]A typical Twitter user has about 200 twitter followers.

3.2.1 Twistor Client

Motivated by our goal that is achieving large recipient anonymity set sizes, we construct a protocol where the sender's original message, encrypted under intended recipient's public key, is relayed through the sender's follower subgraph. For this purpose, we utilize the *Twistor client*, a piece of software being run by follower nodes, which maintains a long-lived HTTPS connection to Twitter's streaming API. Twistor client decrypts the ciphertext it receives to check if the corresponding message is destined for the associated user and reposts the ciphertext to the user's timeline.

3.2.2 Design Challenges

Even though the mentioned design results in a large receiver anonymity set which is desirable for our goal, if left unfettered, this can result in an exponential increase in the number of tweets. However we are not interested in flooding Twitter with messages.

Another challenge is how to ensure the intended recipient falls within the sender's subgraph.

We also need a way to beat Twitter's 140 character limit for the length of tweets.

In what follows, we explain how we overcome these challenges.

Overcoming Twitter's 140 character limit

As already stated in previous chapters, currently Twitter only allows exchange of short messages limited to 140 UTF-8 characters. However, if an image is attached to the Tweet, the maximum limit for the image size would be 3 MB.

ElGamal cryptosystem results in a 2:1 expansion in size from the plaintext to the ciphertext. Normally, a Twistor plaintext consists of the sender's original message appended to the message's checksum[6]. The plaintext is then encrypted using the ElGamal encryption scheme. The resulting ciphertext is then attached to a Twistor-specific hashtag and a TTL indicator. As such, if sender's input message be larger than a certain length, the size of the Twist can easily exceed the 140 character limit.

In order to solve this problem, we stegano-embed larger messages, which are symmetrically encrypted, into an image. We will then only need to encrypt the symmetric key and its associated checksum with the ElGamal

[6]The details are elaborated in the next chapter.

encryption scheme. As the size of the symmetric key is fixed to 16 bytes in our current scheme, this will never exceed the 140 character limit.

Truncating sender's subgraph

In order to avoid flooding Twitter with senders' messages, we attach a counter as a TTL flag to the generated ciphertext. When a client gets a ciphertext, it first decrement the TTL flag by one and reposts the resulting message to associated user's timeline. Additionally, a history of already retweeted messages is kept for some limited time. If Twistor client gets a message that has been posted before or if TTL flag reaches zero, the message is discarded.

Checking for recipient's reachability

We need a way to check if a given node falls in the sender-originated subgraph, which is now truncated as described above. The easiest way to do this is to directly ask twitter or some third party about this information. However, doing so leaks the identity of the intended recipient, which contradicts our purpose of hiding this information. Another option is that the Twistor client crawls the entire follower subgraph down to certain number of levels and then locally checks if this set includes a given node. However, Twitter rate limits its API calls, which makes this approach impractical.

Our proposed solution is to provide this functionality as a separate service, which we will explain in the following section.

3.2.3 Twistor Graph Service

In our proposed design, Twistor clients perform an initial registration with the graph service and declare the list of immediate followers/followees of the corresponding user collectively. The graph server aggregates this information into one large graph G. Upon request or even on certain intervals, the graph server provides clients with the computed subgraph G^* for the associated user. For each user, the nodes in its corresponding subgraph G^* make the set of all reachable nodes down to a certain level.

Definition 1: The Twistor follower graph $G = (V, E)$ is a directed graph of vertices V and edges E, where V and E represent Twistor users and followee/follower relationship among them respectively.

We define the Twistor's follower subgraph for a Twistor user as follows:

Definition 2: The Twistor's follower subgraph for a Twistor user with username U is a subgraph $G^*_U = (V^*, E^*)$ rooted at U induced by fol-

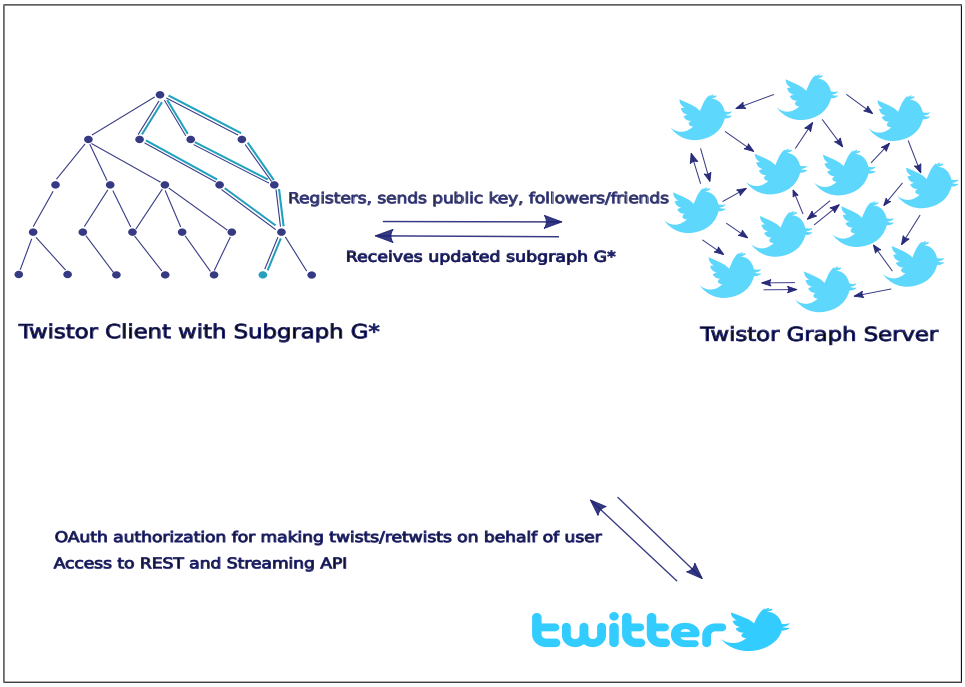


Figure 3.1: An overview of Twistor's system architecture

lowee/follower relationships down to a certain depth l, where $V^* \subseteq V$ and $E^* \subseteq E$.

When an update occurs in the user's list of followees, this event is received via Twistor client's streaming API, which in turn, sends the update to Twistor's graph server. Thus, the graph server maintains an updated version of G over time.

In the meanwhile, each Twistor client declares its (signed) public key to the graph server upon registration. Although Twistor clients could make use of private information retrieval to query for the public keys of some subset of nodes [18], in our current implemented scheme, associated user's public keys are directly incorporated into G^* vertices.

Figure 3.1 illustrates the interaction between our system components.

Chapter 4

Implementation

Twistor client and graph server are written in around 4500 lines of Python. In this chapter, we explain some implementation details for Twistor.

4.1 Twistor Client

4.1.1 Cryptographic Setting

We use NIST elliptic curve based on prime-order groups for primes of length 256 bits as the underlying algebraic group. We chose ElGamal over elliptic curve P-256 as the asymmetric cryptosystem. We deploy AES[7] with 128 bit keys for symmetric encryption and use SHA1 hash digest for message integrity check.

4.1.2 Twist Generation

When a user U intends to send a recipient-anonymous message through the system, she provides the input message and the final recipient's username into the Twistor client's web interface. Using subgraph G_U^*, the Twistor client checks for reachability of the provided recipient. If the check fails, user is notified. The user can then notify the intended recipient out of band to randomly follow one of the nodes in G_U^* in order to make subsequent anonymous communications possible. Otherwise, if the recipient was proved reachable, input message is encrypted under recipient's public key[8] and posted to the Twitter timeline of user U. As previously stated, larger messages are stegano-embedded into an image and then posted to the user's Twitter timeline.

More precisely, for larger messages, a hybrid encryption scheme is used: the input message is encrypted with AES-128, appended with HMAC message authentication code and finally embedded into an image. The AES-128 symmetric key (or the message in case of smaller input messages), is

[7] In cipher block chaining (CBC) mode.

[8] Please note that G^* nodes also incorporate the public key of the associated node.

appended with the message checksum[9]. The resulting message is then encrypted with the recipient's public key. Encrypted message together with the accompanying image (if existing) are posted as an status update to the user's timeline.

A status post generated by Twistor, which we name a *twist*, is accompanied by a hashtag marking the status as being generated by a Twistor client. Additionally, as explained in the previous chapter, a TTL indicator is attached as a clear-text to the twist hashtag. Each client receiving a twist would decrement the TTL indicator by one so as to stop propagation of the twist after certain level.

4.1.3 Reposting a Twist

Twistor clients are constantly listening for new Twists from the corresponding user's followees. When a followee posts a new status, the Twistor client checks for the presence of Twistor hashtag and verifies if the status is a valid Twistor ciphertext. Upon detection of a twist, the Twistor client decrypts the ciphertext and verifies its checksum in order to ascertain whether the twist is intended for the corresponding user.

If the checksum is correct and the twist does not include an image, accompanying plaintext is the sender-originated message which was originally intended for the user. If the twist is accompanied by an image, the Twistor client keeps the plaintext as an AES-128 key and continues with decoding the ciphertext embedded into the image. After decoding the ciphertext and verifying its integrity using HMAC, the client decrypts the ciphertext using the AES-128 symmetric key in order to obtain the original sender's plaintext message.

If hash of the twist was found in a locally maintained dictionary or if the TTL flag was zero, the ciphertext is not reposted any further. Otherwise, Twistor client decrements the TTL indicator by one. The resulting twist, in which we call *retwist*, together with the accompanying image (if any) is posted as an status update to the corresponding user's timeline.

4.2 Graph Server

The multithreaded graph server maintains the most up-to-date graph G. When a Twistor client initially registers with the graph server and declares

[9]We use the first few bytes of the message's SHA1 hash digest as the checksum.

the list of immediate neighbours for the corresponding user U, it receives the subgraph G^*_U.

Later on, as users follow and unfollow other nodes, the vertices and edges in G and G^*_U get updated in the graph server. As a result, subsequent requests of the subgraph, only involves sending the updates as opposed to the entire subgraph to Twistor client. In our implemention, sending incremental updates from the graph server is also possible via pub/sub rather than on demand.

Chapter 5

Evaluation

There exists a body of quantitative research that have analyzed the Twitter social graph in terms of its follower-following topology, its connected components and the distribution of information propagation within each component [28], [29], [30].

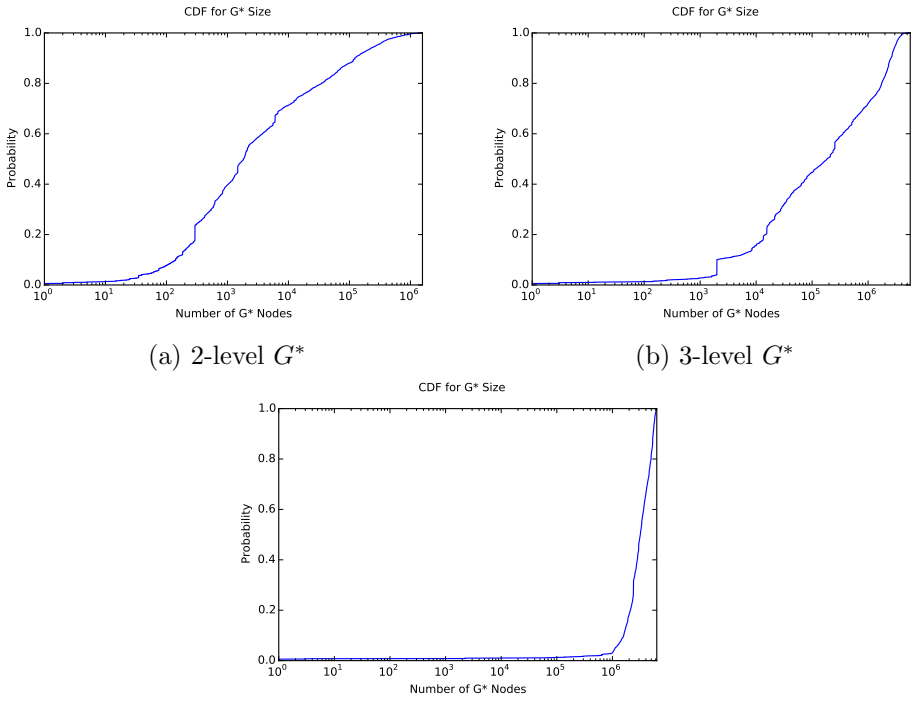
In this section, we provide results for evaluating our system. We have studied the following parameters in our evaluation:

- Number of levels that the ciphertext needs to be relayed, so that the receiver anonymity set size is in the order of hundred of thousands.
- Number of reposts performed by intermediate relays in order to communicate a message between sender and receiver.
- The overall delay in sending the message to the recipient.
- Amount of memory consumption for storing follower subgraph G^*.

It is noteworthy to mention that in practice, where some followers may not use Twistor, fixing the same number of levels that a ciphertext is relayed for all users, results in an unsatisfactorily low number of participating relays and hence a low anonymity set size. Thus, in order to maintain these numbers large enough, the Twistor graph server can tune an appropriate level for a given user's G^* subgraph, so as to obtain the desired anonymity set size. Using this approach, different users might end up having a different number of levels in their corresponding G^* subgraph.

We performed our evaluation on a Twitter graph dataset of 2011 [10] [27]. The dataset was composed of over 11 million users and over 85 million social relations. We conducted our analysis on a random population of 5000 users. All tests are performed on OS X 10.10 with 3GHz Intel i7 and 16GB of RAM.

[10] Mined in 2011 by Arizona State University Data and Machine learning laboratory as a representative sample of the twitter group.



(a) 2-level G^*

(b) 3-level G^*

(c) 4-level G^*

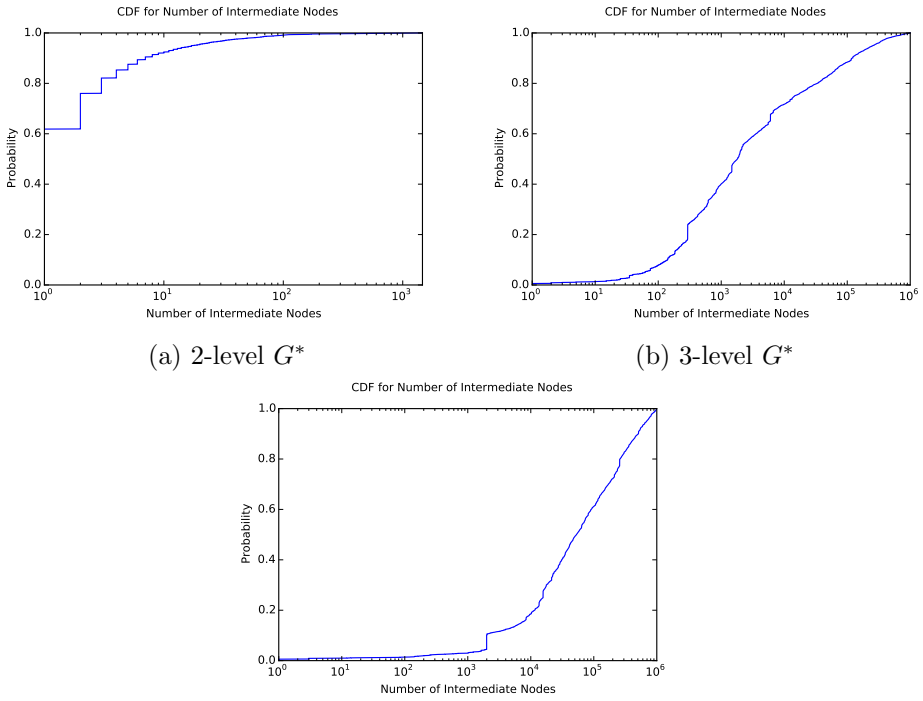
Figure 5.1: CDF for size of G^* for varying number of levels

5.0.1 Size of G^* Subgraph

As already stated in previous chapters, we are aiming for receiver anonymity sets in the order of hundreds of thousands. For this purpose, the G^* subgraph for a typical user needs to include at least that many nodes. We computed G^* rooted at all 5,000 nodes in our sample population for varying levels down the graph.

We found out that for most users, a three-level G^* was sufficient to achieve our desired anonymity set size.

Figure 5.1 shows the cumulative distribution function (CDF) for the number of nodes for varying G^* levels. As shown in the figure, for the three-level G^* only 15% of users in the sample population had less than 10,000 nodes and 60% of users had more than 100,000 nodes in their G^*. The maximum and average G^* size was around 5,000,000 and 700,000 respectively. For a two-level G^* however, only 15% of user had more than 100,000 nodes in their respective G^*.



(a) 2-level G^*

(b) 3-level G^*

(c) 4-level G^*

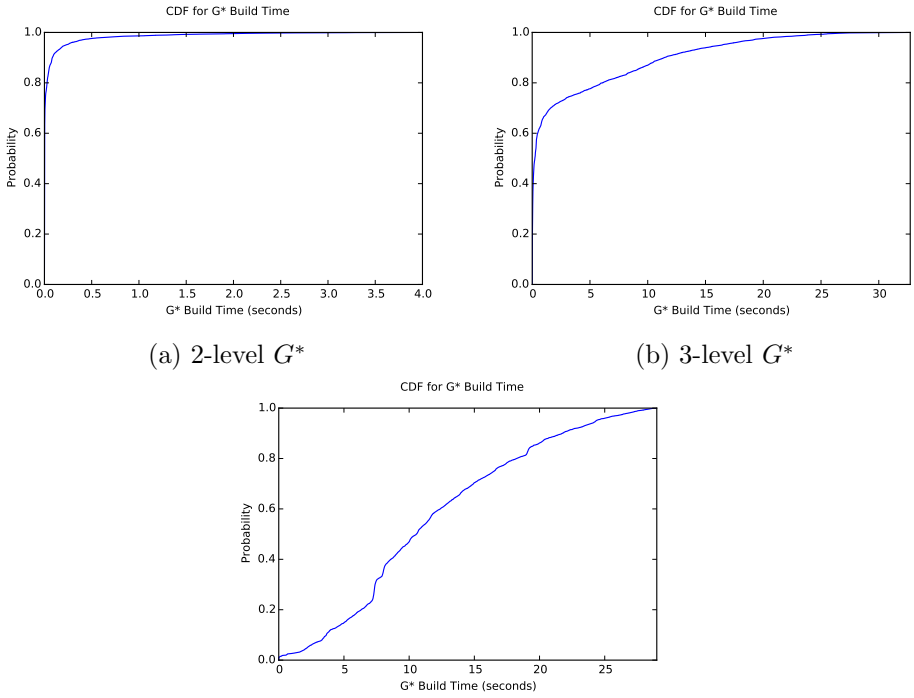
Figure 5.2: CDF for number of reposts of a twist for varying number of levels

5.0.2 Number of Reposts of a Twist

We found out that number of reposts made by intermediate nodes is in fact much less than the size of receiver anonymity set in which we obtain. As illustrated in Figure 5.2, in the three-level G^*, 70% of users in the sample population have fewer number of reposts than 10,000 and for only 10% of users is the number of reposts more than 100,000.

5.0.3 Build Time of G^* Subgraph

We computed the time taken to build the G^* subgraph; depicted in Figure 5.3. According to the results, for 80% of users, three-level G^* build time was at most 5 seconds and the maximum build time was only 32 seconds.



(a) 2-level G^*

(b) 3-level G^*

(c) 4-level G^*

Figure 5.3: CDF for G^* build time for varying number of levels

5.0.4 Memory Consumption and Initialization

We measured the amount of memory consumption for storing G^* nodes in the client. We found out that it took less than 7MB to store an average G^* size of 700,000 nodes. When sending a G^* subgraph to a Twistor client, the server also incorporates the public keys of individual nodes in the response. As it takes 80B to store each public key used in our system, the overall memory consumption for storing G^* subgraph in the client is around 61MB. Assuming 10 Mb/s broadband connection, it takes less than a minute to send the entire G^* subgraph to a Twistor client (61 MB/1.25 MB/s = 48.8). Note that all subsequent interaction with the graph server only involves sending incremental updates (For the nodes being added to or deleted from the G^* subgraph). As a result, the mentioned delay is only a one-time cost.

5.0.5 Twitter API versus Cryptographic Costs

As already stated in previous chapters, we used ElGamal over Elliptic curve of prime-order P256. We calculated the time taken for performing cryptographic operations and determined that encryption/decryption costs are only around 5 milliseconds.

We measured the time taken for posting and reading a twist via Twitter API calls. We found out that Twitter API calls are the dominating costs and in the experiments performed on our fake Twitter accounts, the numbers varied from 400 milliseconds to 1.5 seconds for each repost. This resulted in an end-to-end latency of between 2 to 7 seconds.

Chapter 6

Related Work

6.1 Anonymity through Social Graphs

Twistor employs the Twitter follower graph for routing receiver-anonymous communications. There exist some other works that use social graphs as the backbone of anonymity networks. The social graph is then used for forwarding users' traffic to remote communicating parties. However, as far as we are aware of, these proposals have not resulted in practical adoption and have all targeted sender anonymity.

Drac [4] is the first system in this respect. Drac leverages peers in the social network to relay traffic in a system providing anonymity and unobservability. In order to hide connection and disconnection events, building circuits for anonymous communications are performed in separate epochs, using random walk in the social graph. During each epoch, users randomly select a friend as the first hop and then iteratively extend their circuit via random friends of friends as chosen by the predecessor hop.

In Pisces [5], a system for anonymous web browsing and instant messaging, a random walk in the users' social graph is augmented by a reciprocal neighbour policy. The reciprocal neighbour policy is for punishment of malicious nodes whom try to exclude benign nodes in their declared routing tables. For enforcing the policy and identification of cheaters, each node's current signed list of contacts gets distributed using Whanau distributed hash table. The nodes periodically download and check for presence of several conflicting neighbour lists.

In Dynamix [6], the authors extend theoretical findings for building anonymous communications over static social graphs to that of dynamic graphs; which captures churn normally imposed by joins and leaves in social networks.

VirtualFriendship [7], provides profile-request anonymity and communication unlinkability in a centralized online social network by routing communications outside the social network. For this purpose, trusted routing friends are used that redirect received requests using an external anonymity network to other routing friends. On the other side, the routing friend ver-

ifies authorization token for the request and sends back social network's response.

6.2 Increasing Anonymity Set Size

Another property of Twistor is increasing receiver anonymity set size. Using nodes in the follower graph as relays results in an exponential increase in the size of the receiver anonymity set. Some other works have pursued a similar approach in order to increase sender anonymity set size.

Some designs [8], [9], [10] leverage casual web surfers to increase anonymity set size.

[8], presents a protocol that exploits normal web browsing activities of ordinary users for creating an anonymous overlay network. An anonymous user's browsing activity gets hidden among all other web users contacting the same set of servers, which in turn, participate in the sender anonymity set. The authors propose a Chaumian Mixnet variant, where each Mix node in the system serve a CGI script. When an onion-encrypted message is supplied as an HTTP POST request to a node, the script gets called and emulates the behaviour of a regular Mix, i.e., decrypts the message and looks at the headers to determine further processing. When ordinary users visit webpages served by websites containing a specially crafted banner advert (which includes a frame containing JavaScript created by a node), their browsers execute the JavaScript code returned by the node. This way, ordinary users get involved in transferring the messages to another node.

In [9], the authors introduce a JavaScript framework, called ConScript, which is made up of websites cooperating for enabling anonymous communications. It works by encouraging casual web users visiting cooperative websites to participate in anonymous communications. The cooperative websites serve JavaScript code, instructing the browser to act as a client by creating and submitting encrypted dummy messages into the underlying anonymity system. This provides cover traffic for other users of the system who wish to send real messages by using a browser plugin. In that case, when the user visits a cooperative website, the browser plugin simply intercepts outgoing messages from the browser and replaces dummy message with user's encrypted real message.

6.3 Anonymity through Multicast

Twistor uses Twitter's followers graph as an application-level multicast tree for relaying anonymous messages. Multicast schemes have been used by a large body of previous works in order to achieve sender/receiver anonymity.

[11] is the first protocol that employs (IP-level) multicast routing in order to provide receiver anonymity.

In [12], the authors propose a method for providing receiver anonymity that uses multicast together with a cryptographic primitive called incomparable public keys. Using the suggested primitive, the receiver generates many anonymous identities and many unique public keys, all being associated with the same private key. Upon receiving a ciphertext, all members of the multicast group try decrypting the message and ignore it in case of decryption failure.

[13], presents a protocol for mutual-anonymous communications by using an application-level multicast tree overlay network. Sender anonymity is achieved by making every node in the overlay network participate in relaying messages. Receiver anonymity is achieved by peers setting up their filters to get messages they are interested in. The tree is restructured from time to time so the nodes' parent, siblings and friends change dynamically.

In [14], the authors employ a subscription tree for enabling anonymous communication. In order to publish content anonymously within a group, the publisher sends an anonymous message to the group's root by using Crowds-like [15] probabilistic routing. The group's root then forwards the content along the tree. Nodes willing to receive some content anonymously should subscribe through a random set of proxy nodes; where the last node eventually joins the multicast tree. The published content is then routed anonymously to the receiver. As other nodes should forward and receive content on behalf of interested subscribers, this provides plausible deniability for all members.

P5 [16], introduces an scalable P2P anonymity protocol for satisfying sender-, receiver-, and sender-receiver anonymity. Scalability of the system is achieved through creating a hierarchy of progressively smaller broadcast groups; each level of hierarchy providing different level of anonymity and communication efficiency. Users in the hierarchy are grouped together into broadcast groups. When a message is sent to a broadcast group, it is also forwarded to all groups with an ancestor-descendent relationship with that group.

Chapter 7

Conclusion

Anonymous communications aim to protect communications' privacy within the Internet, where privacy of online users is easily threatened by adversarial breaches. Achieving high degree of anonymity, i.e., large anonymity set sizes, requires a large number of active participants and hence a great amount of cover traffic.

In this thesis, we presented Twistor, a practical anonymity system being built as an overlay on top of Twitter's existing publish-subscribe infrastructure. Twistor leverages the reachability provided by a cascade of tweets and retweets on Twitter to achieve large receiver-anonymity sets. Twistor relies on a special property of ElGamal encryption scheme whereby a ciphertext does not reveal the public key used to produce it.

Twistor's communication cost consists of one tweet operation per Twistor relay. Processing cost includes one elliptic curve decryption operation and one AES decryption, which is linear with the number of twistor relays in the sender's subgraph. Except for being compliant with its normal functionality, we don't put any extra trust assumption on Twitter. As long as relays falling on the sender's chosen path do not misbehave, the message is guaranteed to reach its final anonymous destination.

Bibliography

[1] Andreas Pfitzmann, and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management, v0.34. Accessible through: http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf, 2010.

[2] Andrei Serjantov, and George Danezis. Towards an information theoretic metric for anonymity, In: Proc. Privacy Enhancing Technologies(PETS), 2003.

[3] Claudia Diaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity, In: Proc. Privacy Enhancing Technologies(PETS), 2003.

[4] George Danezis, Claudia Diaz, Carmela Troncoso, and Ben Laurie. Drac: an architecture for anonymous low-volume communications, In: Proc. Privacy Enhancing Technologies(PETS), 2010.

[5] Prateek Mittal, Matthew Wright, and Nikita Borisov. Pisces: Anonymous communication using social networks, In: Proc. Network and Distributed System Security Symposium(NDSS), 2013.

[6] Abedelaziz Mohaisen, and Yongdae Kim. Dynamix: anonymity on dynamic social structures, In: Proc. ACM symposium on Information, computer and communications security(AsiaCCS), 2013.

[7] Filipe Beato, Marco Conti, Bart Preneel, and Dario Vettore. VirtualFriendship: hiding interactions on online social networks, In: Proc. Communications and Network Security(CNS), 2014.

[8] Matthias Bauer. New covert channels in HTTP: adding unwitting Web browsers to anonymity sets. In Proc. Workshop on Privacy in the Electronic Society(WPES), 2003.

[9] Henry Corrigan-Gibbs, and Bryan Ford. Conscript your friends into larger anonymity sets with JavaScript, In Proc. Workshop on Privacy in the Electronic Society(WPES), 2013.

[10] Volker Roth, Benjamin Gldenring, Eleanor Rieffel, Sven Dietrich, and Lars Ries. A secure submission system for online whistleblowing platforms. In Proc. Financial Cryptography and Data Security(FC), 2013.

[11] Brian Neil Levine, and Clay Shields. Hordes: a multicast based protocol for anonymity. Journal of Computer Security 10, no. 3 : 213-240, 2002.

[12] Brent R. Waters, Edward W. Felten, and Amit Sahai. Receiver anonymity via incomparable public keys. In Proc. Computer and Communications Security(CCS), 2003.

[13] Ying Wang, and Partha Dasgupta. Anonymous communications on the internet. In Proc. Computer and Communication Security(CCS), 2005.

[14] Alan Mislove, Gaurav Oberoi, Ansley Post, Charles Reis, Peter Druschel, and Dan S. Wallach. AP3: Cooperative, decentralized anonymous communication. In Proc. ACM SIGOPS European workshop, 2004.

[15] Michael K. Reiter, and Aviel D. Rubin. Anonymous web transactions with crowds. In Communications of the ACM, 1999.

[16] Rob Sherwood, Bobby Bhattacharjee, and Aravind Srinivasan. P5: A Protocol for scalable anonymous communication. In Proc. IEEE Security and Privacy(S&P) Symposium, 2002.

[17] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Advances in Cryptology(ASIACRYPT), 2001.

[18] Prateek Mittal, Femi G. Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: Scalable anonymous communication using private information retrieval. In: Proc. USENIX Security Symposium, 2011.

[19] http://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/, August 2015.

[20] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In: Proc. USENIX Security Symposium, 2004.

[21] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In Proc. IEEE Security and Privacy(S&P) Symposium, 2003.

[22] Ulf Mller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster protocol–version 2. Draft, http://freehaven.net/anonbib/cache/mixmaster-spec.txt, 2003.

[23] Stevens Le Blond, David Choffnes, Wenxuan Zhou, Peter Druschel, Hitesh Ballani, and Paul Francis. Towards efficient traffic-analysis resistant anonymity networks. In Proc. ACM Special Interest Group on Data Communication(SIGCOMM), 2013.

[24] Hsu-Chun Hsiao, TH-J. Kim, Adrian Perrig, Akimasa Yamada, Samuel C. Nelson, Marco Gruteser, and Wei Meng. LAP: Lightweight anonymity and privacy. In Proc. IEEE Security and Privacy(S&P) Symposium, 2012.

[25] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, Cornell University, 2003.

[26] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: making strong anonymity scale. In Proc. USENIX Symposium on Operating Systems Design and Implementation(OSDI), 2012.

[27] Reza Zafarani and Huan Liu. Social Computing Data Repository at ASU [http://socialcomputing.asu.edu]. Tempe, AZ: Arizona State University, School of Computing, Informatics and Decision Systems Engineering, 2009 (updated in 2011).

[28] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a social network or a news media? In Proc. ACM international conference on World Wide Web(WWW), 2010.

[29] Kristina Lerman, and Rumi Ghosh. Information Contagion: An Empirical Study of the Spread of News on Digg and Twitter Social Networks. In Proc. International Conference on Weblogs and Social Media(ICWSM), 2010.

[30] Maksym Gabielkov, Ashwin Rao, and Arnaud Legout. Studying social networks at scale: Macroscopic anatomy of the twitter social graph. In

ACM international conference on Measurement and Modeling of Computer Systems(Sigmetrics), 2014.