StarClique: Guaranteeing User Privacy in Social Networks Against Intersection Attacks

Krishna P. N. Puttaswamy, Alessandra Sala, and Ben Y. Zhao
Computer Science Department, University of California at Santa Barbara
{krishnap, alessandra, ravenben}@cs.ucsb.edu

ABSTRACT

Building on the popularity of online social networks (OSNs) such as Facebook, social content-sharing applications allow users to form communities around shared interests. Millions of users worldwide use them to share recommendations on everything from music and books to resources on the web. However, their increasing popularity is beginning to attract the attention of malicious attackers. As social network credentials become valued targets of phishing attacks and social worms, attackers look to leverage compromised accounts for further financial gain.

In this paper, we analyze the state of privacy protection in social content-sharing applications, describe effective privacy attacks against today's social networks, and propose anonymization techniques to protect users. We show that simple protection mechanisms such as anonymizing shared data can still leave users open to *social intersection attacks*, where a small number of compromised users can identify the originators of shared content. Modeling this as a graph anonymization problem, we propose to provide users with k-anonymity privacy guarantees by augmenting the social graph with "latent edges." We identify *StarClique*, a locally minimal graph structure required for users to attain k-anonymity, where at worst, a user is identified as one of k possible contributors of a data object. We prove the correctness of our approach using analysis. Finally, using experiments driven by traces from the del.icio.us social bookmark site, we demonstrate the practicality and effectiveness of our approach on real-world systems.

Categories and Subject Descriptors

H.3.5 [**Information Systems**]: Online Information Services—*Data sharing, Web-based services*; K.6.5 [**Computing Milieux**]: Security and Protection—*Invasive software, Unauthorized access*

General Terms

Security, Design, Measurement

Keywords

Online Social Networks, Privacy, Intersection Attack, k-anonymity, Graph Anonymization

1. INTRODUCTION

Building on the popularity of online social networks (OSNs) like Facebook, social content-sharing applications allow users to form communities around shared interests. Millions of users worldwide use them to share recommendations on several resources on the web. Facebook application platform alone hosts nearly 52,000 applications [26], many of them provide recommendations between friends for everything from music and books to restaurants, websites, and travel recommendations [1, 2]. Given the success of the social application model, we envision a future where social input can dramatically improve the effectiveness of Internet applications. For example, users can share browsing histories to help avoid malicious web sites, or share web search results to improve the quality of searches [19].

While the benefits of these applications are unquestionable, they can potentially expose their users to significant compromises in privacy by revealing private information about their users. This is particularly a cause for concern given recent reports of attackers trying to compromise social network accounts using both Phishing attacks [3, 28] and social worms [12, 25]. In addition, recent studies have shown that social network user credentials are a primary target of botnets, and thousands of social network user accounts have been stolen by active Internet botnets [10, 27]. In fact, analysis of the Torpig botnet [27] shows that five of the top ten domains targeted for user credentials were social networks. Malware authors can use a compromised account to stealthily monitor a user's friends' activity over a long period of time to gather sensitive information. This data can later be used to customize spam to specific users to increase the effectiveness of spam attacks [13].

Defending against these client-side attacks is a challenging problem. Our analysis shows that traditional anonymization techniques are insufficient, and a mathematically rigorous solution is necessary. For example, simply removing the identities of users from the data they contribute cannot protect user privacy – using just two compromised accounts in a user's neighborhood, an attacker can perform a *social intersection attack* and identify the source of shared data with very high accuracy. More compromised accounts leads to improved accuracy. This attack is highly effective in social content sharing applications, and uses only data available inside the applications, making it nearly impossible to detect.

The goal of this paper is to improve user privacy against such client-side attacks and make it impossible for attackers to definitively identify the source of a piece of shared data. Intuitively, our approach "augments" the structure of the social network with selective friendship edges we call "latent edges," so that each user attains guaranteed privacy in the form of k-anonymity. This means any attacker(s) can at best identify the source of a data object as a member of a group of k users [29]. We model this problem as a social

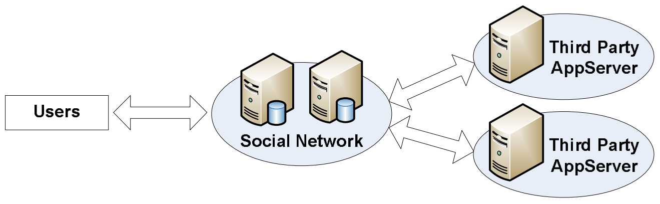

Figure 1: **A typical social network architecture: third party apps. are hosted on remote servers, and are accessed via the social network.**

graph anonymization problem, where users are protected by "latent" edges between her and her socially-close (two or three-hop) neighbors. Our objectives are to provide *provable k-anonymity* for application users, and to do so using a *minimal* number of *socially-close* latent edges. The resulting system is tunable, so OSN operators can use parameters to navigate the tradeoff between different levels of guaranteed privacy and the associated costs of new latent edges.

This paper makes four key contributions: First, we analyze the privacy problem in today's social content sharing applications, and identify the *social intersection attack* in Section 2. Using real datasets, we show that this attack can be highly effective on today's social networks using only a small number of compromised accounts. We also show that this attack cannot be addressed by prior graph anonymization techniques in Section 3. Second, we reduce this privacy problem to a graph anonymization problem in Section 4, and describe a graph evolution approach to modify the social graph to protect user privacy. We identify a graph structure called *StarClique* that guarantees k-anonymity. Deleting any edge from this structure invalidates the k-anonymity property provided by this structure. We also present several optimizations to reduce the number of latent edges added to the social graph. Third, we analyze our solution in Section 5 to prove its privacy and minimality properties, and bound the number of latent edges added. Finally in Section 6, we evaluate our design using real social graphs from several online social networks, and use a detailed case study of the del.icio.us social bookmark service to demonstrate the feasibility of our approach.

2. THREATS AND POTENTIAL DEFENSES

We begin by examining the issue of privacy in social content-sharing applications. To start, we present the standard online social network (OSN) architecture and describe a sample content-sharing application. We then show how compromised users (also referred to as attackers) can launch a *social intersection attack* to identify the owner of shared content, and evaluate its effectiveness on today's social networks. Finally, we discuss several potential solutions, their tradeoffs, and the intuition behind our chosen approach.

2.1 OSNs and Application Background

Figure 1 depicts the architecture of a typical OSN like Facebook, which supports third-party applications run on remote servers. The social graph and user data are stored at the OSN site in a cluster or a "cloud." Third party applications run on their own servers using the API provided by the OSN, store and process application content locally, but interact with users through the OSN. Facebook Application Platform and OpenSocial are two popular examples of platforms with this architecture. The threats to privacy we identify in this paper are common to these centralized architectures, as well as distributed architectures used by Tribler [24], FTN [11], and SocialSearch [19]. However, given that centralized architectures are highly prevalent today, with millions of users, we focus only on the centralized architecture in this paper. We leave extending this work to the distributed architectures for future work.

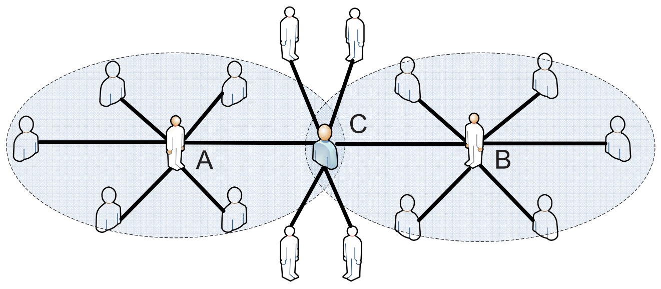

Figure 2: **The social interaction attack: User C has 6 friends in the social network. Compromised friends A and B perform the attack by intersecting their respective social circles, yielding C.**

Social Network	Crawl Date	% of the Graph	Source
del.icio.us	Dec. 2008	6.4	our crawlers
Facebook NY Network	March 2008	45.0	[30]
Flickr	Jan 2007	26.9	[20]
LiveJournal	Dec 2006	95.4	[20]
MySpace	Oct 2006	0.08	[6]
Orkut	Oct 2006	11.3	[20]
Youtube	Jan 2007	unknown	[20]

Table 1: **Statistics of the Social Network Datasets.**

While users in this application model are vulnerable to both compromised users as well as malicious third-party applications, we will focus on user-level attacks in this paper. Verifying the trustworthiness of third party applications is a challenging problem that requires both technical and non-technical solutions. One effort to address this is the Facebook verified application program, which attempts to authenticate and "verify" applications for proper data access by hand [14]. In contrast, just two malicious or compromised users can launch a client-side attack passively with minimal start-up costs, and is much more difficult to detect.

Example application. To illustrate the threats to user privacy in current social applications, we consider the example of a social web-reputation application where users share ratings of the safety/trustworthiness of web sites they visit and warn their friends against potential phishing or malware sites. Users install a plug-in in their browser that stores the URLs they visit in this social application. Each user's browsing history is stored on a third party server. Using this data, application users can cooperatively judge the credibility of a site based on how many of their friends use the site often, and discover similar websites. For example, when a user visits a new site $www.airtickets.com$, the local browser plugin queries the application server for this URL, and returns the number of friends that regularly use this website, and any posted warnings about the website.

Recent events have shown that a user's web browsing habits can reveal extremely personal information about them [8]. Therefore, users wary of revealing too much private information might hesitate to adopt applications like this. To encourage user participation, current applications adopt the standard approach of just removing the contributing user's identity from any shared data. But as we will show next, this simple approach can be easily circumvented by more intelligent privacy attacks such as the social intersection attack.

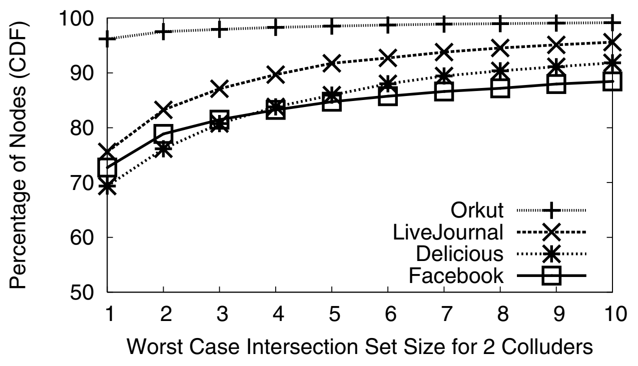

Figure 3: **Worst case impact of intersection attack when *two* compromised users intersect their local graph.**

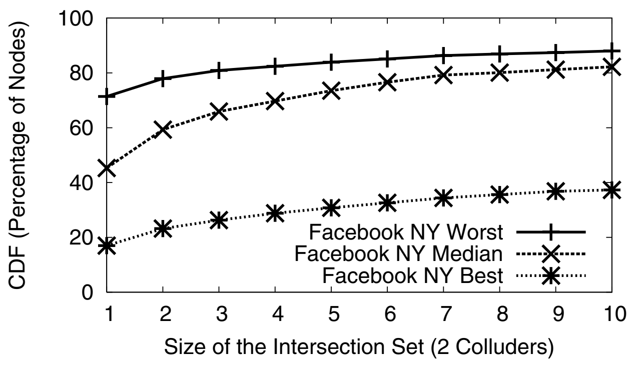

Figure 4: **Distribution of the intersection attack results when *two* compromised users attack Facebook.**

2.2 The Social Intersection Attack

A *social intersection attack* can identify the original owner of anonymized shared data objects. It can be used by compromised users, for example, to find out about the web browsing habits of a common friend. In this attack, two or more compromised users in nearby social circles can periodically perform an intersection of their friend lists, as well as of the shared content they obtain from the social application. The common content observed by all attackers is likely to have come from a common friend. There might be false positives for popular content, *e.g.* different friends might have all visited [*www.google.com*], but for uncommon content, this technique will identify the owner with high accuracy.

Figure 2 shows an example of this attack. Compromised friends A and B are attacking user C. In the URL reputation sharing application, A and B both input the same URL to the application, and perform an intersection of their results and the local graph topology. Since C is the only friend they share, common content (URLs) can be attributed to C.

2.2.1 Impact of the Intersection Attack

There are three key properties of this attack: First, it is *application independent* in that it only relies on the social network structure. Second, users can perform this attack *passively* by sending regular application queries to the target without fear of detection. Third, this attack requires only two attackers and becomes more powerful with additional attackers. Supernodes with high degree in the social graph, *i.e.* very popular users with many friends, are extremely vulnerable. Their privacy can be compromised if any small fraction of their friends turn into compromised users (or attackers).

A natural question to ask next is: *how vulnerable are users to this intersection attack in social networks today?* We answer this question by analyzing real social graph topologies from seven popular OSNs: del.icio.us, MySpace, Facebook, Flickr, LiveJournal, Orkut, and YouTube. We use traces from prior measurement studies [6, 20, 22], our prior work [30], and newly crawled data from del.icio.us (Table 1).

We run an experiment to evaluate each node's vulnerability to this attack if some of her friends are compromised. For each node x with degree ≥ 2, we list its one-hop friends. We then take every possible pair of her friends f_a and f_b, and compute the number of common friends f_a and f_b share. If f_a and f_b perform a social intersection attack, this number indicates the size of the intersection set containing x. For each of our social network graphs, we choose 10,000 random nodes and perform this test on each node. For each user, we take the minimum intersection set size from all potential attacks. We plot the distribution of this value for all 10K users in Figure 3. We see that across all social networks, nearly 70% of the users have a worst case size of 1. This means that, for these users, there exists a subset of only two friends that can uniquely identify this user via intersection attack.

In Figure 4, we plot the minimum intersection set size (worst case), the best case, and median values for the Facebook graph. Roughly half of all Facebook users are likely to be uniquely identified (value of 1) if two of their random friends are compromised. These results are consistent across all seven networks, but we plot only Facebook for brevity.

2.3 Potential Defenses and Shortcomings

We now consider several potential approaches to defend user privacy against social intersection attacks. First, since this attack does not rely on knowledge about the user identity, removing identities from shared content is ineffective. A second option is for the applications to return data only from a random selection of one-hop friends for each query, instead of data from all one-hop friends. The compromised users do not know which random friends are selected by the application, and hence cannot exactly match data to users. However, this approach hurts application functionality: To protect user privacy, a significant amount of data cannot be shared between friends.

A third option is for the applications to *dilute* the data returned to users by returning data from a random set of two or three-hop friends, in addition to the data from one-hop friends, in response to each query. This technique is also ineffective, since attackers can submit the same query multiple times and associate the common data across queries with the one-hop friends that are always common – thus eliminating the cover traffic from two or three-hop friends. Instead, a user's friends-of-friends must be used with the same frequency as her one-hop friends. This, in fact, leads to our chosen approach: Alter the social graph to provide provable privacy guarantees for each user by selectively adding "latent edges" that connect her to her friends-of-friends.

Note that there is a fundamental tension between privacy protection and additional content sharing overhead across latent edges. At one extreme, applications may return shared content from all users to each user, effectively creating social links between all users. This provides maximum privacy by hiding each user among the entire population, but completely removes any social "relevance" of shared data, and imposes significant load on the application servers. The other extreme is to only share content from a user's direct friends, which provides no privacy. To manage this tradeoff, we adapt a solution based on k-anonymity – a tunable approach to privacy first introduced in the database community [4, 5, 29]. In this context, we define k-anonymity to mean the property that the owner of a data object can at worst be identified as one out of a group of k users, each of which is equally likely to be its owner. This provides provably strong privacy, reduces the overheads from adding too many links to the social graph, and allows the system to trade off privacy and overheads using the parameter k.

3. RELATED WORK

k-Anonymity in Databases. k-Anonymity was first introduced in order to generalize entries in a table with sensitive data such that each tuple in the modified table has at least $k - 1$ other tuples that are identical [29]. Several papers have used this concept of k-anonymity since then [5,16,18,23] to provide protection to privacy-sensitive user records while publicly releasing the dataset. The process of k-anonymization of a database is a NP-Hard problem [29], and hence approximate k-anonymity algorithms have been proposed before [5]. However, the primary issue in applying these techniques to the problem in this paper is that the attacker we consider is "online," within the system, and can gain some inside information about the system. But the work on k-anonymity assumes an attacker that attempts to de-anonymize the data after it is publicly released.

Social Network Anonymization. Recently, several social graph anonymization algorithms are proposed to enable public release of social graphs without compromising user privacy [7, 9, 17, 31]. The main goal here is to prevent attackers from identifying a user or a link between users based on the graph structure. There are, however, some key differences that set apart our work. First, in the graph anonymization problem also, similar to the work on databases, the attacker is outside the system. In this paper, we consider a stronger attacker that is a participant of the network (or online), and can relate content received locally from the neighbors in the network. Second, the definition of privacy breach is different in the two cases. In graph anonymization, a user privacy is breached if either a user is identified in the anonymized graph, or a link between two users is established. Our goal, however, is to prevent attackers from linking the data transmitted by applications with the users. Given the abundance of the application data as well as the social graph, it is more challenging to provide anonymity guarantees. Finally, the solutions proposed by prior graph anonymization work [9, 31] provide global privacy properties (as in, create k identical neighborhoods, or k identical-degree nodes in the graph, etc.). These global properties do not ensure that each node in the network has sufficient degree to defend against the intersection attack. For example, the graph in the Figure 5 is the result of applying the solution from the work on neighborhood anonymization [31] to a social graph – there are at least two identical neighborhoods in the graph. While this graph is 2-anonymous against neighborhood[1] attacks, it does not protect against intersection attacks: Any two colluding neighbors of any node in the graph can successfully perform an intersection attack. This example can be extended such that it provides k-anonymity against neighborhood attacks for arbitrary k but still any node remains vulnerable to intersection attack by any two of its neighbors.

Social Network Measurement Studies. Recent measurement studies [6, 20, 22] crawled various social networks and analyzed the social graph properties. Facebook study [30] analyzed the difference between the social graph, and the user interaction graph. Facebook application study [22] analyzed the graph structure of social applications. We thank the authors of these studies for providing the traces.

[1]A vertex v's neighborhood is the induced subgraph of v's one-hop neighbors.

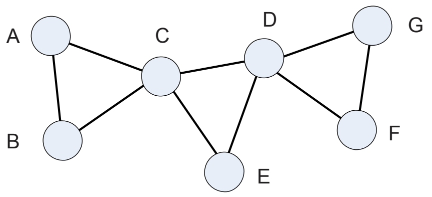

Figure 5: An example anonymized graph to defend against neighborhood attacks [31], which is still vulnerable to intersection attacks.

4. GRAPH EVOLUTION FOR PRIVACY

Naturally formed social graphs tend to exhibit power-law degree distributions and high skew in node connectivity. Local clustering is limited, and the lack of common friends makes users vulnerable to the social intersection attack. Our solution to this problem is to "evolve" the graph by adding "privacy buddies" to users such that all users have k-anonymity, for some value of k chosen by the OSN operator. Adding these buddies creates *latent edges* between buddies and users. The real and latent edges together provide k-anonymity. The evolved graph with privacy guarantees is used by the applications to transfer data between users, but this evolved graph is never revealed to the users directly. As a result, attackers do not know the list of friends sending data to them and cannot identify the exact source of the data they receive. The only change existing social networks need to do, to use our solution, is to evolve the graph, and send the evolved graph to the application servers instead of the real social graph.

4.1 Assumptions, Goals and Attacker Model

We next list our assumptions, goals, and the attacker model for this paper. And later we describe detailed design of our solution followed by our algorithms and optimizations.

4.1.1 Assumptions

We make two simple assumptions in our design. First, we assume that the OSN operators and third-party application servers are secured by the owners and do not compromise their users' privacy. These sites have significant financial incentives to keep their service secure: To attract and retain their users. The end users, on the other hand, may be lax in applying security patches and hence be compromised due to various malware attacks. Second, our privacy mechanisms are irrelevant if user identities can be deduced directly from shared data. So we assume that all data is scrubbed to remove identifiable user information. This scrubbing can happen before the data leaves a trusted endpoint. Similarly, we assume that the attackers cannot cross correlate application data with out-of-band information to identify its owner, as was done in recent NetFlix privacy attack [21].

4.1.2 Goals

Our goal is to provide three key properties to *all* users in the network irrespective of their social connectivity.

Provable k-Anonymity. We aim to provide *k-anonymity* to social application users. k-anonymity provides source anonymity and the data receiver cannot tell the source even with social intersection attack. Formally, k-anonymity is:

DEFINITION 1. *The system provides k-anonymity to the source (x) of an event ξ, if the probability that the attackers assign for x to be the source of ξ is less or equal to $1/k$. In other words, the attackers suspect at least k different nodes to be the likely sources of ξ, with equal probability.*

Low Overhead. It is necessary to add *minimal* number of latent edges to reduce the additional overhead on the social infrastructure due to processing and data transfer of cover traffic along the latent edges.

Preserve Relevance of Cover Traffic. The latent edges added should connect nodes that are *close in social distance*, so that the cover traffic is still relevant to users. Nodes that are farther apart have fewer "similarities" in interests and connecting them might send highly irrelevant data to users.

4.1.3 Attacker Model

In the social application setting we consider, we assume the following attacker model:

1. A fraction (p) of the one-hop friends of a given user x are compromised. They can work both independently, and in collusion to compromise honest user's privacy.
2. The attackers have the entire social graph. They know their local graph, and can crawl the rest of the graph.
3. We assume that only the attackers within one hop from a user x collude together to break x's privacy via passive intersection attacks.

This is a stronger attack model compared to prior work on graph anonymization [9, 17, 31] as the attackers here use both the application data and the social graph to attack. In addition, passive attacks are harder to detect compared to active attacks. For example, an active attacker can delete all but one of her friends, and assign the new data received to that friend. However, such attacks will be easily detected. Finally, note that the actual number of malicious nodes around a node x depends on its degree (d_x) and the fraction p. We use f to represent the number of malicious neighbors of a node throughout the paper, but f is node specific, and $f = \lceil d_x p \rceil$.

4.2 Graph Evolution

Here we present the details of our social graph evolution mechanisms. First we introduce the StarClique graph structure, and then present a simple algorithm to evolve the graph.

4.2.1 Overview of the Evolution Algorithm

We propose a heuristic that builds a structure around each node by adding latent edges. This structure is called StarClique, and is shown in Figure 6. StarClique is mainly responsible for achieving the k-anonymity property. We build this structure around each node by going through all the nodes in the network one at a time. Since this structure is built locally around each node, it provides several opportunities to optimize the evolution algorithm to reduce the total number of latent edges added to the entire network. We explore several such optimizations to reduce the overall number of latent edges added to the graph.

4.2.2 StarClique Structure

Figure 6 depicts the StarClique structure and its formation. There are two main parts in the structure: The portion to the right (in the central sub-figure) is the clique, and the portion to the left is called the Star. Let the node that is evolved be x, and let f be the number of attackers around x. A StarClique is built around x using its neighborhood nodes. The clique for x consists of x along with its $(k + f - 1)$ neighbors, that together form a $(k + f)$-clique. The Star consists of the one-hop friends of x that do not belong to the clique. Each member of the Star is connected with all $(k + f - 1)$ members of the clique.

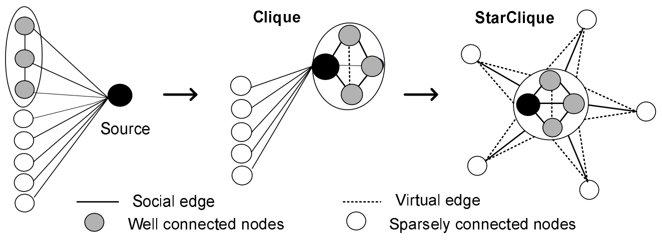

Figure 6: The graph evolution process for a node. The node first selects a subset of its neighbors. Then it builds a clique with the members of this subset. Finally, it connects the clique members with all the non-clique members in the neighborhood. Latent or virtual edges are added in the process.

StarClique provides two key properties: (a) StarClique provides provable k-anonymity to node x against f one-hop colluders, (b) StarClique is a locally minimal structure in the sense that no edges can be removed while still providing k-anonymity against f one-hop colluders. We formally prove the above properties in §5, but we describe the intuition behind these two properties next.

k-Anonymity Intuition. There are f malicious nodes around x. After building StarClique, these f nodes are either all in the clique, or all of them are in the Star, or are distributed between Star and clique. If all of them are in the clique, since the clique size is $(k+f)$, the attackers all see k common friends. If all f attackers are in the Star, then they all see at least k common friends. If they are distributed, between Star and clique, we still have the intersection of all the attackers one-hop neighborhood size to be k. Hence, k-anonymity is preserved in all the three cases (proof in §5).

Edge Minimality Intuition. Deleting any one edge in a StarClique eliminates the k-anonymity property above. For example, if one of the edges in the clique is removed, then we can always find a combination of f nodes in the clique whose one-hop neighborhood intersection gives $(k - 1)$ common nodes instead of k in the clique. Since we do not know the attacker a priori, we need to protect against every possible combination of f nodes to achieve k-anonymity. Along similar lines, we can show that removing an edge between Star and clique doesn't preserve k-anonymity of StarClique (proof in §5).

Function of Latent Edges. The latent edges added during evolution are treated just like real edges between friends by the social applications. These latent edges, however, are not visible to the users. As a result, the users do not know exactly which of the new friends help them achieve privacy. Even if the new friends are revealed to the users, k-anonymity is still preserved. However, not revealing the latent edges further enhances privacy as attackers have to first guess the right latent friends protecting a given node, and then attack a node. Hence we hide these latent friends from the users.

4.2.3 Evolution Algorithm

With this background in mind, now we describe the evolution algorithm. The evolution algorithm works on one node (say x) at a time, and it works in three steps. The first step in evolving a node x is to identify the closest neighborhood of x that has at least

Algorithm 1 Evolution Algorithm: evolves the input graph g to produce an evolved graph g'.

```
Graph $g'$ = Evolve_Graph (Graph $g$)
1: $g' = g$
2: /* Copy the original graph g into the evolved graph g' */
3: for all $x \in V$ do
4:     $N(x) = \emptyset;\ \ i = 1$
5:     while $|N(x)| < (k + f - 1)$ do
6:         $N(x) = N(x) \cup g.neighbors(x, i)$
7:         /* Neighborhood is selected in the original graph */
8:         $i = i + 1$
9:     end while
10:    $C$ = select $(k + f - 1)$ random nodes from $N(x)$
11:    $C = C \cup \{x\}$
12:    $Build\_Clique(g', C)$
13:    /* Edges added in g' to build the clique structure around x */
14:    $Build\_Star(g', N(x), C)$
15:    /* Edges added in g' to build the Star structure around x */
16: end for
17: Return $g'$
```

Algorithm 2 Optimized Evolution Algorithm: evolution algorithm annotated with the optimizations.

```
Graph $g'$ = Evolve_Graph_Optimized (Graph $g$)
1: $g' = g$
2: $S(x) = \{V'$ nodes sorted in the decreasing order of degree$\}$
3: [Applying Optimization 3: Ordered Evolution above]
4: for all $x \in S$ in decreasing order of degree do
5:     $N(x) = g.neighbors(x, 1);\ \ i = 1$
6:     [$g$ (instead of $g'$) is used above to handle the side-effects of Edge Reuse]
7:     while $|N(x)| < (k + f - 1)$ do
8:         $N(x) = N(x) \cup g'.neighbors(x, i)$
9:         [Applying Optimization 2: Edge Reuse above]
10:        if $|N(x)| > (k + f - 1)$ then
11:            [Apply Optimization 4: Limit to k Friends here]
12:        end if
13:        $i = i + 1$
14:    end while
15:    $C = Select\_Clique(N(x))$
16:    [Applying Optimization 1: Select Clique above]
17:    $C = C \cup \{x\}$
18:    $Build\_Clique(g')$
19:    $Build\_Star(g', N(x), C)$
20: end for
21: Return $g'$
```

$(k + f - 1)$ nodes in it. Evolution starts with one-hop neighborhood, and moves to two-hop, and beyond until it gets a big enough neighborhood. Selecting nearest neighbors first ensures that the privacy buddies are closer in social distance. The second step of evolution is to select a subset of $(k+f-1)$ nodes from the neighborhood obtained from the first step, and build a $(k+f)$-clique out of them by adding latent edges. These clique members are selected at random in this simple algorithm. The final step of evolution is to connect all the members of x's neighborhood that do not belong to the clique with all the members of the clique by adding latent edges. This process forms a structure as shown in Figure 6. This structure around x provides k-anonymity to the node x. We analyze the security properties of this structure in more detail later. The $Evolve\_Graph$ algorithm, described here, is presented in Algorithm 1, and the notations used are listed in the Table 2.

4.3 Optimizing the Evolution Algorithm

As we mentioned before, our approach uses a local graph augmenting approach to protect users from malicious nodes that try to link data to a particular user. This local approach may add a large number of latent edges to the graph. Therefore, we propose several optimizations to reduce the number of latent links added to the entire graph. These optimizations significantly reduce the number of new latent edges added to the graph during evolution. We apply our optimizations to $Evolve\_Graph$, and present an optimized algorithm called $Optimized\_Evolve\_Graph$ (shown in Algorithm 2) that is annotated to show where each optimization is applied.

Optimization 1: Select Clique. *Choosing the nodes that share the maximum number of friends with x as the clique members during the construction of StarClique around x decreases the number of new latent edges added.* While selecting the clique members from the neighborhood, choosing the most well-connected $(k + f - 1)$ nodes, instead of random nodes, reduces the latent edges added significantly. $Select\_Clique$ function in the Algorithm 2 implements this optimization, where the well-connected nodes are chosen based on the number of friends shared between the nodes in the neighborhood and x. This selection leads to clique reuse in the neighborhood of x, reducing the new edges added. Similarly, applying this optimization iteratively to every node in the network ($\forall\ node\ x \in V$) leads to significant reduction in the overall number of new latent edges added in the network.

Formally, the most well-connected $(k + f - 1)$ nodes used in the optimization are the $(k + f - 1)$ nodes x_i, in $N_1(x)$, with the highest number of common nodes between $N_1(x)$ and $N_1(x_i)$, where $N_1()$ includes only the nodes in the one-hop neighborhood of a node.

Optimization 2: Edge Reuse. *Sharing latent edges among the neighboring nodes reduces the total number of latent edges introduced.* Before evolving a node x, this optimization considers x's most recent and evolved state that includes the latent edges of x, instead of x's connectivity in the original graph. This reduces the new edges added: As evolution progresses, more and more latent edges are added, and the connectivity of nodes around x increases. This means that x's neighborhood is more connected than in the original graph and hence the number of new edges necessary to evolve x is reduced significantly. Extending this optimization to the whole network iteratively further reduces the total number of new edges added to the network during evolution. This optimization implies that Algorithm 1 should use the evolved graph g' in the loop instead of the original graph g.

Side-effects of Edge Reuse. This optimization should be carefully applied. When x is about to be evolved, if its new degree is $> k + f - 1$ but its original degree was $< k + f - 1$, then only the original neighbors and $l = (k + f - |g.\text{neighbors}(x, 1)|)$ (that are necessary to get a set of $k + f - 1$ nodes) new neighbors need to be considered. Similarly, if the new degree (after evolving some of the neighbors) is $> k + f - 1$ and the old degree was also $\geq k + f - 1$, then only the original degree needs to be considered.

Optimization 3: Ordered Evolution. *Constructing the StarClique structure from high degree nodes to low degree nodes reduces the overall number of latent edges added in the evolved graph.* The intuition behind this optimization is that when a supernode is evolved, many of its neighbors can reuse the latent edges added to evolve the supernode. Since supernodes have a large number of neighbors, many neighbors can benefit from the edges added to evolve the supernodes.

Optimization 4: Limit to k Friends. *Limiting the size of the extended neighborhood of a node during evolution reduces the number of latent edges added.* If a node x has $< k + f - 1$ nodes,

$G = (V, E)$	Graph definition
x, y, z	Nodes $\in G$
$N(x)$	Set of nodes in the neighborhood of the node x
$g.neighbors(x, i)$	Set of all neighbors of x at most i hop away in g
d_x	Degree of the node x
p	Fraction of malicious one-hop neighbors
f	Number of malicious neighbors of x ($f = \lceil d_x p \rceil$)
C	Subset of nodes in $N(x)$

Table 2: Notations used in this paper.

the unoptimized algorithm considers the larger neighborhood incrementally one hop at a time. It is quite likely that when the neighborhood increases by one hop, the neighborhood size goes significantly beyond $k + f - 1$. However, all the nodes in this y-hop are not necessary to provide k-anonymity: We need only $l = (k + f - |g.\text{neighbors}(x, 1)|)$ additional nodes. Thus, we select only the l most well-connected nodes from outside the one-hop neighborhood in this optimization.

Optimized Evolve Graph. The evolution process is depicted in Figure 6, and the pseudo-code for $Optimized\_Evolve\_Graph$ is shown in Algorithm 2. In this algorithm, first, the nodes are sorted by their degree, and evolved in the order of their degree, starting from the highest. Second, evolution is applied on the evolved graph repeatedly – this applies the edge reuse optimization. Indeed, we use the original graph g to get the original degrees, and the evolved graph g' to maximize the number of reused edges during the neighbor selection. This is necessary to handle the side-effects of edge reuse optimization, as described before. When the node has less than $k + f - 1$ friends, its neighborhood in the evolved graph is selected. We apply the Limit to k Friends optimization at this step. Finally, Select Clique optimization is applied in this optimized algorithm while choosing the clique members out of the neighbors. StarClique is built for each evolved node as in $Evolve\_Graph$.

5. ANONYMITY ANALYSIS

This section has three main parts. First, we introduce formal notations and identify the conditions under which k-anonymity is preserved (see Theorem 1). Second, we present the properties of StarClique that are necessary to provide k-anonymity. Finally, we quantify the total number of new edges added while building StarClique around a node.

5.1 Formal Notations and k-Anonymity

We represent the social network as a graph, $G = (V, E)$, where each user is mapped to a unique vertex $\in V$ and the friend relationship between two users x and $y \in V$ is represented as an edge (x, y). V is the set of all vertices, and E is the set of all edges. Each edge is undirected, as it represents the friendship between two user. And an undirected edge $(x, y) \in E$ is equivalent to two directed edges $\overrightarrow{(x, y)}$ and $\overrightarrow{(y, x)}$, where $\overrightarrow{(x, y)}$ represents that x is a friend of y and $\overrightarrow{(y, x)}$ represents that y is a friend of x.

In our model, each edge also represents the flow of information – the flow of events between users in the social network. Therefore, we can formally define:

- (x, y) is an undirected edge where x and y are both senders and receivers of events;
- $\overrightarrow{(x, y)}$ is a directed edge where x is the sender and y is the receiver of events.

An event is usually a message generated by the application, such as a bookmark event in del.icio.us, that needs to be delivered to

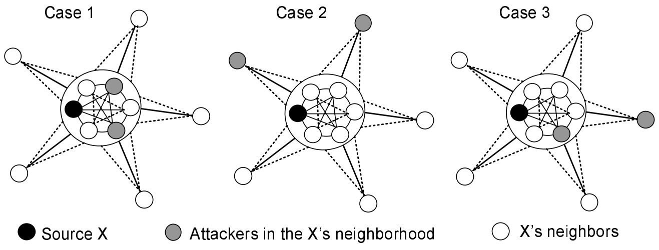

Figure 7: The attackers may be located in any position in StarClique, we identify the three cases that cover the possible positions.

the source's one-hop neighborhood. We aim to disseminate events while providing provable privacy that the user's identity is protected no matter which f one-hop neighbors collude (the compromised friend accounts). In order to prove the guarantees of our solution, we introduce the following definitions: (a) let x be a node $\in V$ and d_x its degree; (b) let f ($>= 2$) be the number of one-hop neighbors of x that are colluding against x; (c) let $v_x =< x_1, x_2, ..., x_f >$ be a vector of randomly selected nodes from x's one-hop neighborhood, and finally (d) let $B = \{v_{x_i} | v_{x_i} =< x_1^i, x_2^i, ..., x_f^i > \ and \ x_j^i \in N(x)\}$ be the set of all possible combinations of f neighbors of x, which has size $|B| = \binom{d_x}{f}$. According to our Definition 1 and the attacker model, a node's k-anonymity is preserved as long as there are at least k common friends in all possible intersections of the f colluding one-hop neighbor subsets.

THEOREM 1. *Given a user x in the social graph, if $\cap_{j=1}^{f} N(x_j^i) \geq k \quad \forall v_{x_i} \in B$ then x's k-anonymity is preserved.*

PROOF. Let x be the source of a particular message sent to all nodes in x's neighborhood. Assuming that a random set of f nodes $\{x_1, x_2, ..., x_f\} \in N(x)$ that are receiving the message are malicious, the probability that x is recognized as the real sender has to be less or equal to $\frac{1}{k}$. In order to guarantee this bounded probability, at least k nodes around the f attackers have to look as possible sources. This means that the f attackers have to share k common neighbors. Formally, let $y_1, y_2, .., y_n$ be n nodes which $\in N(x_1) \cap N(x_2) \cap, ..., \cap N(x_f)$. So, if $|\{y_1, y_2, .., y_n\}| \geq k$ (note that $\forall i \ and \ j, \ y_i \neq x_j$), then there are at least k nodes which could have sent the message and x's identity is covered among those nodes $\in \{y_1, y_2, .., y_n\}$.

On the other hand, if $|\{y_1, y_2, .., y_n\}| < k$, then the real source is identified with a probability $> \frac{1}{k}$, which violates the k-anonymity requirement. Since we do not know the colluding neighbors of x a priori, any combination of x's neighbors of size f should have an intersection neighbor size greater than or equal to k to preserve k-anonymity. This can be formalized as $\forall v_{x_i} = \{x_1^i, x_2^i, ..., x_f^i\}$, with $i = 1, 2, ..., \binom{d_x}{f}, \cap_{j=1}^{f} N(x_j^i) \geq k$. □

5.2 Privacy via the StarClique Structure

The evolution algorithm protects privacy by building StarClique around nodes. The first step to prove the k-anonymity property of evolution is to identify the necessary conditions that the StarClique structure has to satisfy in order to provide k-anonymity for a particular source (x). StarClique (Figure 6) is constructed around x in two steps as follows: (1) **Clique**: Build a clique C of $k + f - 1$ nodes $\in N(x)$ around x. Note that the edges in C are *bidirectional*. (2) **Star**: The remaining nodes $\in N(x) \backslash C$ are connected to a total of $k+f-1$ nodes in the clique[2]. Each edge in this step is directed *from* the clique nodes *to* the Star nodes. Directed edges are necessary only to prove the structure minimality. We next prove, in Theorem2, that StarClique guarantees k-anonymity, and no edge can be removed from it to satisfy this property.

THEOREM 2. *The StarClique is a locally minimal structure providing k-anonymity against f one-hop colluding neighbors: It always guarantees this property and, if any one edge is removed from it, it cannot guarantee the property in general.*

PROOF. There are three possible locations that f attackers can occupy in StarClique, as shown in Figure 7. For each of these, we prove k-anonymity and the minimality property.

Case 1: f colluding nodes $\{y_1, y_2, ..., y_f\}$ are all in the clique. Since the clique size is exactly $k+f$, each attacker is connected to the remaining $k+f-1$ nodes in the clique and when they collude, there are k remaining non-malicious nodes in the clique. This implies that k nodes appear to be the sender with probability $\frac{1}{k}$, which means that the k-anonymity property is preserved. Let us assume that one of the f colluding nodes has $k+f-2$ edges in the clique instead of $k+f-1$ (i.e. it is connected to $k-1$ non-malicious nodes instead of k). In this case $\cap_{j=1}^{f} N(y_j) = k-1$ which does not satisfy the k-anonymity property.

Case 2: f attackers $\{y_1, y_2, ..., y_f\}$ are in the Star. Each node in the star has $k+f-1$ edges coming from the clique by construction even though there are $k+f$ nodes in the clique. Having f attackers in the star means that, in the extreme case, each attacker can exclude one different nodes in the clique, and so at most f nodes in total. As a result, the final intersection set size of the attackers is: $\cap_{i=1}^{f} N(y_i) = k$, which preserves k-anonymity. Now, let's assume that one of the f colluding nodes had $k+f-2$ edges coming from the clique instead of $k+f-1$. In this setting, the extreme case intersection set size of the attackers is: $\cap_{i=1}^{f} N(y_i) = k-1$. This does not satisfy k-anonymity, which means no edge can be cut, proving the minimality.

Case 3: Some of the f attackers are in the clique and some are in the Star. Formally, let ϵ be $\in]0,1[$, and let $\lceil \epsilon f \rceil$ be the number of attackers in the clique and $\lfloor (1-\epsilon) f \rfloor$ the number of attackers in the Star. Because $\lceil \epsilon f \rceil$ attackers are part of the clique, the possible number of sources is $k+f-\lceil \epsilon f \rceil$. There are $\lfloor (1-\epsilon) f \rfloor$ other attackers in the Star that are connected to $k+f-1$ nodes in the clique instead of $k+f$, as before. The attackers in the Star can at most eliminate one node each out of the possible $k+f-\lceil \epsilon f \rceil$ sources. Putting together the results from the two sets of attackers, the remaining possible sources are $k+f-\lceil \epsilon f \rceil - \lfloor (1-\epsilon) f \rfloor = k$. As a result, k-anonymity is preserved. To prove minimality, note that, if we cut an edge either from a node in the clique or a node in the Star, we break the k-anonymity property. The f attackers can be anywhere in the clique or in the Star. So if the removed edge is incident on one of the attackers, then $\cap_{i=1}^{f} N(y_i) = k+f-1-\lceil \epsilon f \rceil - \lfloor (1-\epsilon) f \rfloor = k-1 < k$, which breaks k-anonymity, thus proving minimality. □

5.3 Edges Introduced by StarClique

Here we bound the number of new edges added to the evolved graph using the worst connectivity among a source's neighborhood. In this analysis we treat an undirected latent edge to be equivalent to two directed latent edges.

THEOREM 3. *In the worst case, the number of directed latent edges introduced for a node x with degree d_x is less than $max\{(k+f)^2, d_x(k+f)\}$.*

[2]Note that the star members need to be connected with a total of only $k+f-1$ nodes in the clique, and not $k+f$, for minimality.

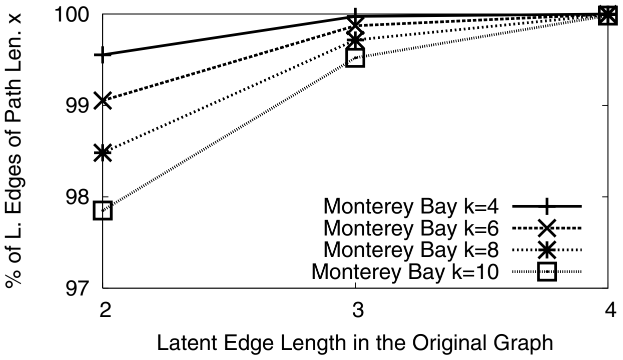

Figure 10: Distribution of latent edge lengths in evolved Facebook Monterey Bay graph as k changes.

PROOF. The proof is divided in two parts as follows:

Case 1: If $d_x < k+f$, we still need each node to be hidden among at least k other possible senders, and hence our algorithm will pick the nodes in its nearest neighborhood which contains at least k nodes before building a clique. Therefore, because a clique of k nodes has $\frac{(k+f)(k+f-1)}{2}$ edges, the maximum number of edges are introduced when the nodes lie on a line topology. Hence, the maximum number is:

$$\frac{(k+f)(k+f-1)}{2} - (k+f-1) =$$

$$(k+f-1)(\frac{k+f}{2} - 1) < (k+f)^2$$

Case 2: If $d_x \geq k+f$, the topological structure that adds the maximal number of latent edges is the *star* structure with the node we are evolving at its center. In the star structure, each neighbor of the central node has no edges to any other nodes. As a result, in order to construct the StarClique structure around a node x with degree d_x, our algorithm adds: $\frac{(k+f)(k+f-1)}{2} - (k+f-1) = (k+f-1)(\frac{k+f}{2} - 1)$ edges to build the clique around x. As each of them is an undirected edge, the total number is $(k+f-1)(k+f-2)$. The remaining $d_x - (k+f-1)$ of x's neighbors need to be connected to $(k+f-2)$ nodes[3], which produces additional $[d_x - (k+f-1)](k+f-2)$ edges. Therefore, the total number of new edges is:

$$(k+f-1)(k+f-2) + [d_x - (k+f-1)](k+f-2) =$$

$$(k+f-2)(d_x) < d_x(k+f). \quad \square$$

6. EVALUATION

We present experimental evaluation of evolution here. First, we describe the experimental setup. Then, we evaluate the impact of the various optimizations in terms of the total number of new edges added to the evolved graph. Finally, we evaluate the impact of evolution on the social graphs.

6.1 Setup

We evaluate evolution using different OSN traces from prior measurement studies [6,20,22,30]. Since the graphs from these studies are very large in size, running evolution on the full graph is very

[3]It is 2 less than $(k+f)$ because each node in the star needs to be connected with $(k+f-1)$ nodes by construction, and each node is already connected to the source x.

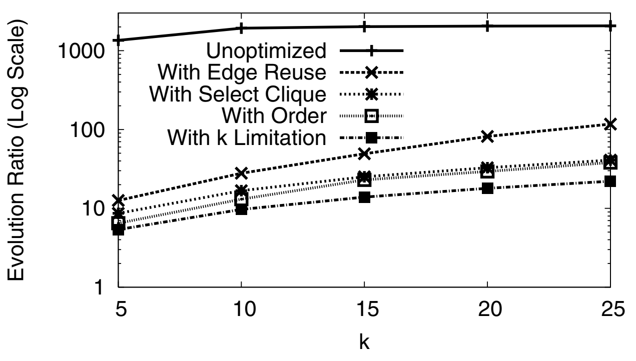

Figure 8: **Impact of optimizations on evolution of Facebook NY graph. One additional optimization added to plots from top to bottom. The plots are ordered in the order of the legends.**

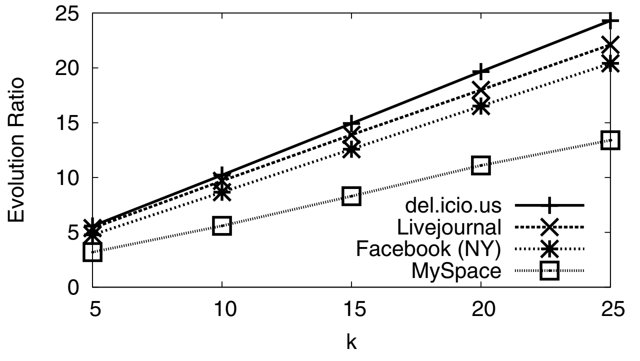

Figure 9: **Ratio of the total number of edges in the evolved graph to the total number of edges in the original graph as k changes.**

resource intensive. As a result, we use a combination of small and large graphs in our evaluation. We use smaller subgraphs to evaluate evolution in detail, to understand the impact of various optimizations and measure evolution's performance on different networks. Then, we use larger graphs from Facebook and del.icio.us to validate evolution's performance on larger datasets.

We used Snowball sampling (or BFS), to sample smaller subgraphs from the social network traces. Snowball sampling is the methodology used in the crawls of many measurement studies [6, 20, 22, 30]. In addition, snowball sample of a certain minimum size is expected to preserve many of the topological properties of the social network [6, 15]. The smaller subgraphs we used were of size 20K nodes. For each experiment on a smaller subgraph, we generated 5 different subgraphs using Snowball sampling starting from different random points in the full network graph, ran evolution on each subgraph 5 times, and present the averaged results.

To complement the results from smaller subgraphs, we use a crawl of the Facebook Monterey Bay network crawl of 260K users, and a del.icio.us crawl with 320K users for a second set of experiments. Given that Facebook had 60 Million users and del.icio.us had 5 Million users during our crawl, the sample sizes of 0.43% and 6.4% (respectively) exceed the minimum portion of the topology that preserves the topological structure of the graph [6, 15]. We use these larger samples to validate our results from the smaller samples.

Finally, we implemented our algorithm using the Python NetworkX module. We used two machines each with 32GB of RAM and Quad-core processors to run our experiments.

6.2 Evolution Algorithm Evaluation

We first present the impact of various optimizations, and then use the optimized algorithm for the rest of evaluation.

Metric: Evolution Ratio. We use the ratio of the total number of edges in the evolved graph to the number of edges in the original graph to evaluate evolution. The lower the ratio, the better the algorithm.

Impact of Optimizations. We use subgraphs from the Facebook New York network to evaluate the impact of our optimizations. Figure 8 plots evolution ratio for different values of k, and $f = 1$, for all four optimizations. The top line is for the unoptimized algorithm, and one optimization is added to the previous as we go down the plots. Figure 8 shows two orders of magnitude improvement in the performance of evolution due to various optimizations. The ratio for a value of $k = 5$ goes down from around 1350 in the unoptimized version to around 5 after applying all optimizations. Similarly, the values for $k = 25$ goes down from 2064 to 22. The main improvement comes from the Edge Reuse optimization, with a relative reduction in the ratio of nearly 10 times. The next major reduction comes from Select Clique with a reduction of nearly 4 times. Optimizations are applied here in one particular order, but we expect the gain from each optimization to remain the same irrespective of the order. We use the fully optimized algorithm for the rest of the evaluation.

Evolution Ratio in Different Graphs. Figure 9 plots the evolution ratio for different networks using the optimized algorithm. For $k = 5$ most networks produce a ratio around 5. However, the ratio for different networks diverges for higher k. The observed ratio mainly depends on the connectivity of the nodes in the graph as measured using clustering coefficient. Myspace, for example, has a large clustering coefficient (nodes are densely connected amongst each other), hence leading to fewer new edges. We confirmed that networks with higher average clustering coefficient appear below networks with lower coefficient in Figure 9.

The evolution ratio approaches the value of k because, to achieve k-anonymity, each node must have a degree of at least k. Since the majority of nodes in our subgraphs have a degree $< k$, to protect these nodes, close to k edges must be added, leading to a increase by k. We ran this test on 20K subgraphs of seven different networks (del.icio.us, Facebook, Flickr, LiveJournal, Myspace, Orkut, and YouTube). As the results were very similar across networks, we present the results for the networks with the lowest ratio (MySpace) and high ratios (del.icio.us, Facebook, LiveJournal).

Impact of Increasing f. In the experiments so far, we explored the impact of the parameter k while keeping f fixed at 1. Next, we explored the impact of larger values of f while keeping the value of k fixed. In these experiments, we found that increasing the value of f increases the evolution ratio linearly, thus leading to an overall evolution ratio proportional to $k + f$. As the result of varying k or f is similar, we do not discuss this further.

Validation Using Larger Graphs. To validate the impact of evolution on larger graphs, we used del.icio.us and Facebook Monterey Bay graphs. Because of the large size of these graphs, and the associated time and memory necessary for evolution, we were restricted to values of $k < 10$. Table 3 summarizes the results from our runs. As expected, we see that the evolution ratio is linear on the value of k in larger graphs as well. Given that these larger graphs are sufficiently large to preserve the topological properties of the social networks (as described before), we believe that the

k	4	6	8	10
del.icio.us	4.15	5.73	7.32	8.9
Monterey Bay	4.14	5.68	7.22	8.76

Table 3: **The variation of evolution ratio with k for large graphs.**

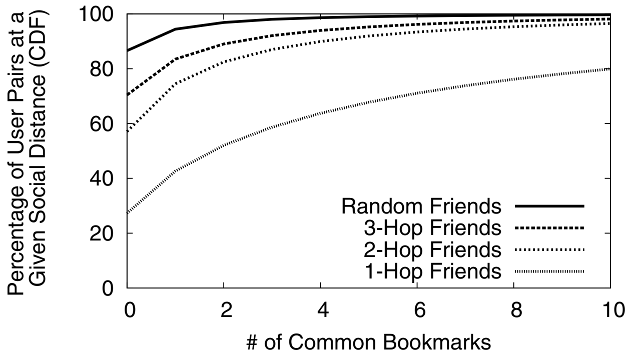

Figure 11: **CDF showing the relationship between the number of common bookmarks between users vs. their social distance.**

evolution ratio resulting from running evolution on the full social graphs to be linear on k.

Social Distance of Privacy Buddies. Next, we validate if evolution chooses privacy buddies from nearby neighbors. We plot the distribution of social distances in the original graph versus latent edges for Monterey Bay in Figure 10. More than 99% of the edges are only two hops away for $k \leq 6$. This number goes down slightly (to 98%) for larger values of k (10). However, most of the chosen privacy buddies are close to the evolved node in social distance. We saw similar results in other networks as well.

6.3 Case Study: del.icio.us

Now we use our del.icio.us crawl to understand the relationship between the relevance of data exchanged between the users and the social distance of the users. Then, we evaluate the application-level impact of evolution using the del.icio.us bookmark data.

6.3.1 del.icio.us Measurement

del.icio.us is a social bookmarking site where users can store bookmarks online, organize them by tags, and share them with friends. The social networking feature lets users find other users with interesting bookmarks, friend them, and obtain updates when new bookmarks are created by their friends. The social network is the primary channel for discovering new content in del.icio.us. As a result, this data is ideal for understanding the relevance of close-distance privacy buddies.

Data Collection Methodology. We implemented a distributed crawler in Python, and crawled del.icio.us from 32 machines at a very slow pace (1 request per second) from 17th of December to 26th of December 2008. We crawled both the social network graph, and all the bookmarks stored by the users over their entire lifetime on del.icio.us. Users have the option to make bookmarks private, and hence we crawled all the publicly viewable bookmarks of the users.

High-Level Statistics. In our crawls, we collected 480K user profiles, 127 million bookmarks in total, with an average around 267 bookmarks per user. This crawl accounts for 6.5% of the total user population, as of December 2008, and 80% of the total bookmarks.

Hop Length	1-Hop	2-Hop	3-Hop	Random
Common Bookmarks	8.45197	1.83435	1.05939	0.316237

Table 4: **Average number of common bookmarks between friends x-hop away. Random represents the test where pairs of users were randomly chosen. Each value is averaged across 1M links (pairs of users).**

k	Orig Graph	3	4	5	6
Bookmarks/Node	5K	21.2K	27.5K	34.2K	39.5K
Bookmark Ratio	1	4.2	5.5	6.8	7.9

Table 5: **The number of bookmarks del.icio.us users receive from their one-hop friends before and after evolution, for different k values.**

6.3.2 Common Content vs. Social Distance

Now we examine the number of bookmarks that friends share (have in common), and its relationship to social distance. Common bookmarks count is a measure of similarity in interests, and will help us to understand the utility of socially-close privacy buddies. Figure 11 shows the distribution of the number of bookmarks shared between one-hop, two-hop, and three-hop friends, and randomly selected pairs of users in the network. Randomly selected pairs are most likely far away in social distance. Figure 11 shows that content sharing between one-hop friends is the highest. As expected, this sharing decreases with increase in the social distance: two-hop friends share much less than one-hop, but better than three-hop. Finally, even three-hop sharing is much better than content sharing between random pairs of users. Table 4 shows the average number of bookmarks users share with their friends for different social distance, which also shows the same trend.

6.3.3 Application-Level Impact

Evolution increases the total number of edges in the network. This increase in edges makes each user in the network send and receive more data along these new edges. We quantify this increased data transfer using the del.icio.us bookmarks as example. Table 5 shows the impact of increase in edges on del.icio.us users in terms of the ratio of total number of bookmarks users receive before and after evolution. Since del.icio.us users get all the one-hop friends' bookmarks, we calculated the number of total bookmarks of the one-hop users to obtain the numbers in the Table. Clearly, the increase in the total number of bookmarks (and hence the server load) is also proportional to the value of k.

In summary, these results (a) clearly support our decision to give preference to nearby nodes as privacy buddies during evolution rather than random nodes, and (b) show that the OSN operators can tune the overhead incurred on the application servers by tuning the value of k.

7. CONCLUSIONS

In this paper, we studied privacy risks involved in sharing data in today's social content-sharing applications due to compromised user accounts. We identify the *social intersection attack*, a low-cost privacy attack that can be used by two or more compromised users to identify the source of shared data objects in all content-sharing applications. It effectively links data objects with their owners relying only the social graph topology and the data shared by the applications. This attack invalidates naive solutions to mitigate privacy risks.

Social networks can provide their users with privacy guarantees in the form of k-anonymity by adding new edges to the social graph. We identify a graph structure we call StarClique, and prove that it is a locally minimal structure providing each user with

k-anonymity. A privacy-conscious OSN provider can build StarCliques around each user, and utilize several optimizations to dramatically reduce the cost of new edges. This type of "graph evolution" is practical for today's social content-sharing networks, and provides sufficient flexibility for OSN operators to make local decisions about the privacy and overhead tradeoff.

Acknowledgments

We thank Christo Wilson, the anonymous reviewers, and our shepherd Augustin Chaintreau for many suggestions to improve the paper. This work is supported by the National Science Foundation under grants IIS-847925, CNS-0916307 and CAREER CNS-0546216. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] Living social. `http://apps.facebook.com/livingsocial`.

[2] Movie recommendataion. `http://apps.facebook.com/apps/application.php?id=2558160538`.

[3] ACOHIDO, B. Phishing attack spreads through facebook. `http://blogs.usatoday.com/technologylive/2009/05/phishing-attack-spreads-through-facebook.html`.

[4] AGGARWAL, G., FEDER, T., KENTHAPADI, K., KHULLER, S., PANIGRAHY, R., THOMAS, D., AND ZHU, A. Achieving anonymity via clustering. In *Proc. of PODS* (2006).

[5] AGGARWAL, G., FEDER, T., KENTHAPADI, K., MOTWANI, R., PANIGRAHY, R., THOMAS, D., AND ZHU, A. Approximation algorithms for k-anonymity. *Journal of Privacy Technology* (2005).

[6] AHN, Y., HAN, S., KWAK, H., MOON, S., AND JEONG, H. Analysis of topological characteristics of huge online social networking services. In *WWW* (2007).

[7] BACKSTROM, L., DWORK, C., AND KLEINBERG, J. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW* (2007).

[8] BARBARO, M., AND ZELLER, T. A face is exposed for AOL searcher no. 4417749, August 2006. NY Times.

[9] HAY, M., MIKLAU, G., JENSEN, D., TOWSELY, D., AND WEIS, P. Resisting structural re-identification in anonymized social networks. In *Proc. of VLDB* (2008).

[10] HOLZ, T., ENGELBERTH, M., AND FREILING, F. Learning More About the Underground Economy: A Case-Study of Keyloggers and Dropzones, 2008. Reihe Informatik. TR-2008-006.

[11] HUANG, Q., WANG, H. J., AND BORISOV, N. Privacy-Preserving Friends Troubleshooting Network. In *Proc. of NDSS* (2005).

[12] JONES, K. C. Facebook expands security tools while combating phishing attack. InformationWeek, May 2009.

[13] KANICH, C., KREIBICH, C., LEVCHENKO, K., ENRIGHT, B., VOELKER, G., PAXSON, V., AND SAVAGE, S. Spamalytics: an empirical analysis of spam marketing conversion. In *Proc. of CCS* (2008).

[14] KINCAID, J. Facebook verified apps now live. Washington Post, May 2009.

[15] LEE, S., KIM, P., AND JEONG, H. Statistical properties of sampled networks. *Physical Review E* (2006).

[16] LEFEVRE, K., DEWITT, D. J., AND RAMAKRISHNAN, R. Incognito: efficient full-domain k-anonymity. In *Proc. of SIGMOD* (New York, NY, USA, 2005), ACM.

[17] LIU, K., AND TERZI, E. Towards identity anonymization on graphs. In *Proc. of SIGMOD* (2008).

[18] MEYERSON, A., AND WILLIAMS, R. On the complexity of optimal k-anonymity. In *Proc. of PODS* (New York, NY, USA, 2004), ACM.

[19] MISLOVE, A., GUMMADI, K. P., AND DRUSCHEL, P. Exploiting social networks for internet search. In *Proc. of HotNets* (2006).

[20] MISLOVE, A., MARCON, M., GUMMADI, K. P., DRUSCHEL, P., AND BHATTACHARJEE, B. Measurement and analysis of online social networks. In *Proc. of IMC* (Oct 2007).

[21] NARAYANAN, A., AND SHMATIKOV, V. Robust de-anonymization of large sparse datasets. In *Proc. of IEEE S&P* (Oakland, CA, 2008).

[22] NAZIR, A., RAZA, S., AND CHUAH, C.-N. Unveiling facebook: A measurement study of social network based applications. In *Proc. of IMC* (2008).

[23] PARK, H., AND SHIM, K. Approximate algorithms for k-anonymity. In *Proc. of SIGMOD* (New York, NY, USA, 2007), ACM.

[24] POUWELSE, J., GARBACKI, P., WANG, J., BAKKER, A., YANG, J., IOSUP, A., EPEMA, D., REINDERS, M., VAN STEEN, M., AND SIPS, H. TRIBLER: a social-based peer-to-peer system. *Concurrency And Computation* (2008).

[25] SELTZER, L. Koobface smacks facebook users. PC Magazine blog, December 2008.

[26] STONE, B. Facebook finally gives apps some love. NY Times Bits Blog, May 20, 2009.

[27] STONE-GROSS, B., COVA, M., CAVALLARO, L., GILBERT, B., SZYDLOWSKI, M., KEMMERER, R., KRUEGEL, C., AND VIGNA, G. Your botnet is my botnet: Analysis of a botnet takeover. In *Proc. of CCS* (2009).

[28] SUDDATH, C. The downside of friends: Facebook's hacking problem. Time, May 2009.

[29] SWEENEY, L. k-Anonymity: A Model for Protecting Privacy. *Intl. Journal of Uncertainty, Fuzziness and Knowledge-based Systems* (2002).

[30] WILSON, C., BOE, B., SALA, A., PUTTASWAMY, K. P. N., AND ZHAO, B. Y. User interactions in social networks and their implications. In *Proc. of EuroSys* (April 2009).

[31] ZHOU, B., AND PEI, J. Preserving privacy in social networks against neighborhood attacks. In *Proc. of ICDE* (2008).