# FriendGuard: A Friend Search Engine with Guaranteed Friend Exposure Degree

Joshua Morris
EECS Department
University of Missouri
Columbia, Missouri
jdm6b3@mail.missouri.edu

Dan Lin
EECS Department
University of Missouri
Columbia, Missouri
lindan@missouri.edu

Anna Squicciarini
Information Science and Technology
Pennsylvania State University
University Park, Pennsylvania
asquicciarini@ist.psu.edu

## ABSTRACT

With the prevalence of online social networking, a large amount of studies have focused on online users' privacy. Existing work has heavily focused on preventing unauthorized access of one's personal information (e.g. locations, posts and photos). Very little research has been devoted into protecting the friend search engine, a service that allows people to explore others' friend lists. Although most friend search engines only disclose a partial view of one's friend list (e.g., $k$ friends) or offer the ability to show all or no friends, attackers may leverage the combined knowledge from views obtained from different queries to gain a much larger social network of a targeted victim, potentially revealing sensitive information of a victim. In this paper, we propose a new friend search engine, namely FriendGuard, which guarantees the degree of friend exposure as set by users. If a user only allows $k$ of his/her friends to be disclosed, our search engine will ensure that any attempts of discovering more friends of this user through querying the user's other friends will be a failure. The key idea underlying our search engine is the construction of a unique sub social network that is capable of satisfying query needs as well as controlling the degree of friend exposure. We have carried out an extensive experimental study and the results demonstrate both efficiency and effectiveness in our approach.

**Keywords:** Privacy Preservation, Friend Search, Social Network

**ACM Reference Format:**
Joshua Morris, Dan Lin, and Anna Squicciarini. 2019. FriendGuard: A Friend Search Engine with Guaranteed Friend Exposure Degree. In *The 24th ACM Symposium on Access Control Models and Technologies (SACMAT '19), June 3–6, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3322431.3325103

## 1 INTRODUCTION

Social networks, such as Facebook and Twitter, are a way of connecting countless people through online sharing and media. One key feature on many social networks is the friend search engine. This engine usually allows users to display the friend list of another user in order to foster new connections, and therefore allow more interactions between people that would otherwise not be connected.

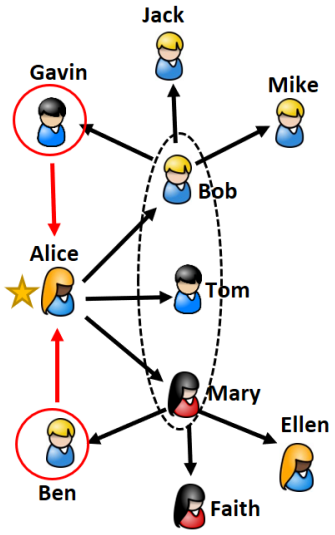



**Figure 1: An Example of Attack Scenario**

Although this is a useful feature to some, a friend engine may help leak unwanted and unnecessary information of one's network, raising privacy concerns. By crawling an online social network through API tools, a few researchers have found that the friend search engine is a lot more vulnerable to privacy breaches than many would expect. Most of a user's friend list can be revealed using a series of carefully designed queries that crawl the user's friends of friends [1, 18]. Figure 1 shows an example of such a vulnerability. User's queried by the friend search engine will return the nodes they point to. In this case, we will focus on *Alice*. From this example, assume *Alice* only wants to show *Bob*, *Tom*, and *Mary*. Instead, *Gavin* and *Ben* want to show her as a connection. If every single node within this network is queried, *Alice* will have her entire friends list shown including *Gavin* and *Ben*. This could similarly be true if *Alice* wished that her friend list was private. This shows that with incompatible settings or random selections, an entire user's friend list can be inferred through the picks of others.

This vulnerability enables attackers to gain intelligence of a target's close friend network, and possibly use this information to gain leverage or obtain sensitive information regarding the targeted individuals. For example, a business man may not want to disclose


*SACMAT '19, June 3–6, 2019, Toronto, ON, Canada*

ACM ISBN 978-1-4503-6753-0/19/06…$15.00
https://doi.org/10.1145/3322431.3325103

all of its social connections to its competitors since that may contain business intelligence; a politician may not want to reveal his relationships with all his potential supporters. Such attacks clearly violate users' privacy expectations. Unfortunately, this is an under looked issue among social networking users. Someone who has created a private network and friend list should not have to worry about their connections being exposed through a work around in the system. When security controls are bypassed, users may feel hesitant to continue using a platform, which would be a loss for both the product owner and the individual user.

In the recent years, there have been a few approaches that aim to preserve users' private social connections against malicious friend searches [6, 12]. However, the latest research shows that they are still vulnerable under collusion-based attacks [8, 9].

In this paper, we propose a new friend search engine, called FriendGuard. FriendGuard maintains the same key function of the previous defense that allows people to view a maximum $k$ friends of a user. Meanwhile, FriendGuard provides strong privacy guarantees. That is, if a user only allows $k$ of his/her friends to be disclosed, our search engine will ensure that any attempts of discovering more friends of this user through querying the user's other friends will be a failure. For example, the previous attack scenario noted by Figure 1 will be avoided if *Gavin* and *Ben* were not reported as the top $k$ friends of *Alice*.

The key idea underlying our search engine is the construction of a sub social network that is capable of satisfying query needs as well as controlling the degree of friend exposure. Specifically, we extract a minimum sub-network from the existing social network so that any individual or collaborative friend search will not disclose information outside this sub-network. Every node in this sub-network only has a desired amount of connections limited by each user's privacy policy. We have developed efficient algorithms to support friend queries in real time. These algorithms include a greedy algorithm that attempts to pick the top friends of a current user, a depth first search algorithm that prioritizes a single pick for each current user along a path, and an optimized method for the depth first search algorithm that picks root nodes for each path by smallest degree in the original online social network. We have also carried out an extensive experimental study and the results demonstrate both efficiency and effectiveness of our approach.

The remaining sections of the paper are organized as follows. Section 2 discusses the related work. Section 3 discusses the overall goal of our paper in relation to the current security vulnerabilities. Section 4 talks about each specific algorithm and how they achieve a more secure friend search engine. Section 5 overviews our implemented algorithm across a variety of tests. Section 6 talks about some of the shortcomings of our solution and other possible problems. Finally, Section 7 concludes the paper.

## 2 RELATED WORK

With the growing concerns of privacy issues incurred during the growth of online social networking, many researchers and service providers have attempted to propose solutions that help provide better privacy protection for users. Most of the efforts are placed on location privacy and information sharing privacy [2, 5, 7, 14–17]. Very little studies have been carried out to preserve unexpected social network exposure caused by the online friend search engine. Several vulnerabilities in the existing friend search engine of Facebook have been pointed out in [1]. Specifically, Bonneau et al. [1] found an attack that is able to discover nearly the entire friend list of a user by conducting an abundant amount of random friend queries using the friend search engine. In [18], Ren et al. point out that attacks that use intersection set computation may reveal more users' friend information if no privacy protection mechanism is in place. Such kind of attacks are especially concerning since users have no control of it and may not even be aware of these vulnerabilities, even though they have properly set up their privacy configurations, e.g., sharing only a small set of their social contacts.

To address the privacy concerns during the friend search, Ren et al. [12] propose a remediation approach that uses a privacy-aware method to verify that the given set does not leak information. In this solution, user A queries the friend search engine for user B. If a friend of A and B, say C, is found, then the friends of C and B can be intersected to find the overlap. This intersection set of B and C will be recommended to A by the friend search engine, based on the findings that many users are willing to accept friend requests to strangers if they have a common connection [11]. By doing this, common connections are needed to gain an accurate estimate of a user's network. While this is useful, it restricts the amount of connections given through a friend search engine to a node that is not inside their common connections. Our approach aims to give controls to individual users, that can specify the exact amount of friends that they would like to disclose through a friend search engine. This would also help to give picks that are outside of a common connection, allowing more diverse sets to be given than this method allows.

Another solution proposed by Li [6] implements a policy that allows users to set a $k$-friend limit to the query results returned by a friend search engine. This solution utilizes a stack method that tracks the queries made, and changes results based on prior returned sets. This algorithm checks if current query results would violate the $k$-friend integrity of the previous searches when performing the current one. If the query violates this, the algorithm will not give the user(s) that would be violated, and instead gives another suitable option. By implementing this method, they are able to give optimal sets as available, but will then switch to more secure results if further queries would cause the $k$-friend policy limit to be violated.

An example of this defense is shown in Table 1 which references Figure 2. Suppose that an attacker queries *Mike* and *Tom*, when there is no defense. The friend search engine gives the nodes specified by there arrows. By adding both results we can see that Tom has friends *Mike*, *Bob*, and *Sarah* violating K=2. When the defense scheme is in place though, the algorithm changes its behavior after giving the results from Mike. Because it knows that giving two new friends for *Tom* would violate his k friend policy, it instead returns *Mike* and *Bob*. This allows the friend search engine to return a set that does not violate *Tom's* k friend policy of 2.

This defense scheme was later discovered by the same researchers to contain a vulnerability. In [8, 9], collusion attacks could be made on this defense scheme to gain more exposure than the $k$ friend limit policy allowed. By pooling information across multiple attackers making queries, queries can bypass the defense method by making

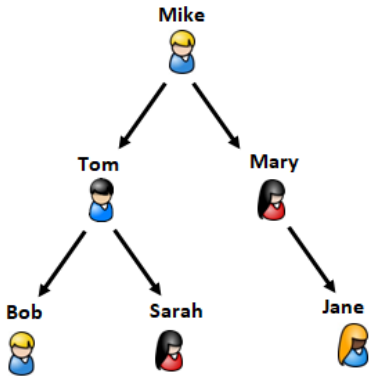



**Figure 2: Sample OSN**

**Table 1: Query Results Comparison**

| User Being Queried (k=2) | *Mike* | *Tom* |
|---|---|---|
| Query results without defense | *Tom, Mary* | *Bob, Sarah* |
| Query results with defense | *Tom, Mary* | *Mike, Bob* |

queries that gain information of a user's friend list not possible if the queries were made by the same attacker. Because this defense uses a stack like system to log their queries, the stack would not remain for another user, even if they were sharing information, because it was not in the scope of their defense.

**Table 2: Collusion Attack**

| User Being Queried (k=2) | *Mike* | *Tom* |
|---|---|---|
| Query results obtained by $1^{st}$ attacker | *Tom, Mary* | *Mike, Bob* |
| Query results obtained by $2^{nd}$ attacker | no query | *Bob, Sarah* |

The example in Table 2 shows how the defense stack is bypassed by a collusion attack. Specifically, when the first attacker queries *Mike's* friends, the friend search engine returns *Tom* and *Mary*. Afterwards, if the first attacker queries about *Tom*, the friend search engine will return *Mike* and *Bob* but not *Sarah* in order to prevent the attacker from knowing more than two friends of *Tom*. However, this defense scheme is not able to prevent two or more attackers who combine their query results. For example, if the second attacker does not query *Mike* but only query *Tom's* friends list, it is possible that the friend search engine returns any two friends of *Tom* which could be *Bob* and *Sarah*. Once these two attackers share their query results, they will know that *Tom* has three friends: *Mike*, *Bob* and *Sarah*. This violates *Tom's* privacy policy whereby he only wants to disclose two friends. While this example is simple, there is a lot that goes into making attacks on a network through collusion to explore most of a user's contact list. Such collusion attack violates users' privacy settings and result in the same overexposure as if there was no defense for this network.

Unlike previous works which relies on runtime privacy checking and query modification, we tackle the problem from a new angle by extracting a sub-network from the online social network to form a robust platform for any friend search engine. We ensure that queries performed on our constructed sub-network will be robust to various kinds of attacks including the newly discovered collusion attacks, and provide a strong guarantee of preserving users' friend exposure degree.

## 3 PROBLEM STATEMENT

In this work, the online social network is modeled as an undirected graph as defined in Definition 1.

Definition 1. ***(Online Social Network or OSN)*** *An online social network is defined as an undirected graph $G(\Xi, R)$, where $\Xi$ is the set of the users in this social network, and $R$ is the set of edges connecting pairs of users who have relationship with each other, i.e., $R = (u_i, u_j)$ where $u_i, u_j \in \Xi$.*

In the online social network, the users can set the number of contacts that they allow the friend search engine to disclose. This privacy setting is called *friend policy* in our paper.

Definition 2. ***(Friend Policy)*** *A friend policy $P(u_i, k_{u_i})$ denotes that user $u_i$ allows no more than $k$ of his/her contacts to be disclosed by a friend search engine.*

Definition 3. ***(Friend Query)*** *Given $P(u_i, k_{u_i})$, a friend query on $u_i$ is denoted as $Q(u_i, k_{u_i})$ which will return a set of 0-$k_{u_i}$ users from $u_i$'s contact list.*

Our proposed approach aims to honor user-defined friend policies by ensuring the friend exposure degree (Definition 5 under the attack model as defined in Definition 4).

Definition 4. ***(Attack Model)*** *Attackers are assumed to be able to query any user through the friend search engine, and may share their query results to gain more knowledge.*

Definition 5. ***(Friend Exposure Degree)*** *Let $u_i$ denote a user on an online social network. User $u_i$'s friend exposure degree (denoted as $\xi_{u_i}$) is the number of friends who will be revealed by aggregated query results through any number of queries on the friend search engine.*

## 4 THE FRIENDGUARD SEARCH ENGINE

As users and regulations define more strict privacy control, a fault in the friend search system that allows attackers to violate these privacy preferences is a big problem. Attackers who obtain a large percentage of a targeted users' friend information could develop an intelligent social-graph of that targeted user [1, 18] to gain leverage. This leads to growing privacy concerns from users, and possibly fault on the social network for not upholding their own privacy rules. To address these issues, we propose a new scheme to protect social connections during the friend search, which is called FriendGuard.

Our FriendGuard scheme takes a dramatically different approach compared to prior works [6, 12]. The FriendGuard will be operated by OSN (online social network) service providers which have the full knowledge of their own social networks. Our main idea is to help OSN providers to mark a sub-network for the friend search engine to constrain the friend exposure for all the users. The sub-network can either be built all at one time, or can be built incrementally as queries to the friend search engine are made. It is worth

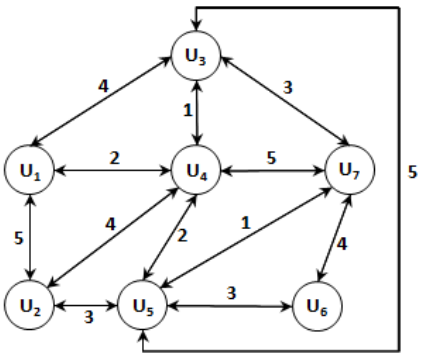



**Figure 3: The Original Online Social Network**

noting that the sub-network does not need extra storage space. Instead, the OSN providers will just need to add a flag to the user links to indicate whether this link belongs to the sub-network and can be used during the friend query.

The FriendGuard scheme can be operated in two modes: (i) unweighted friend selection; (ii) popularity-based friend selection. In the first mode, the FriendGuard will return any $k$ friends of a user being queried without considering how frequently the user interacts with the $k$ friends. In the second mode, the FriendGuard will strive to return the most popular friends while preserving the friend exposure degrees. It is worth noting that it is almost impossible to always return the most popular friends while guaranteeing the friend exposure degree for each user, which is likely to lead to overexposure as shown in the previous examples. In the following subsections, we will discuss the trade-off that needs to be made to ensure the privacy guarantee.

Figure 3 gives an example social network that will be used to compare the results from our proposed two types of friend search engine results. Specifically, Figure 3 shows the social connections among 7 users, whereby the edges between two users indicate the two users are friends of each other, and the value on each edge indicates an aggregated popularity score between the two users. Assume that the OSN providers have their own functions for obtaining the original popularity scores. We will model the connection between two users by using their average popularity score. For example, if user $U_1$ has a popularity score 6 among $U_2$'s friends according to his/her contact frequency with user $U_2$, and user $U_2$ scores 4 among $U_1$'s friends, their aggregated mutual popularity will be simply modeled as (6+4)/2=5 as denoted on the graph. This helps us to consider each connection extracted from the original social network to be bidirectional. That means that if user $A$ is returned as a friend of user $B$ during the query, user $B$ will also be in user $A$'s friend query list. In this way, we eliminate the possible collusion attack as defined by definition 4.

## 4.1 Unweighted Friend Selection

To support the unweighted friend selection, we will construct the sub-network as follows. First, we sort all the users (i.e., nodes in the OSN) in an increasing order to denote which users will be selected first. We have explored three kinds of selection types as presented

**Algorithm 1:** FriendGuard Independent User Selection

```
/* Ξ is a set of nodes within an OSN. This set can
   be an individual node or the total nodes within
   the network. N_i denotes the node selected.   */
Data: ( Ξ )
/* net is the main OSN, whereas sub_net is the FSE
   specific sub-network. Calling
   sub_net[N_i].friends.size() gives the current
   connections of N_i.                           */
for ( N_i ∈ Ξ ) do
    k = N_i.k_friend_policy();
    friends_left = k - sub_net[N_i].friends.size();
    for ( N_j ∈ net ) do
        if ( friends_left < k ) then
            if ( net[N_i][N_j] are friends && N_j != N_i &&
              sub_net[N_j].friends.size() < k ) then
                if ( N_j ∈ sub_net[N_i].friends ) then
                    continue;
                else
                    sub_net[N_i].friends.append[N_j];
                    sub_net[N_j].friends.append[N_i];
                    friends_left -= 1;
                end
            end
        else
            break;
        end
    end
end
```

in the following three algorithms: (1) Independent User Selection; (2) Path Traversal User Selection; (3) Degree Based Path Traversal User Selection.

The Independent User Selection algorithm (Algorithm 1) is the least complex algorithm that could help efficiently build the sub-network, and also allow this network to focus on one user's top picks at a time. This algorithm is based off the assumption that picking each user's top friends will result in the most optimal sub-network for the FSE (friend search engine). This greedy strategy picks the maximum amount of connections for each current node that is selected. Each selection made is a bidirectional pick. This means that if user A chooses user B to be a connection, user B must also choose user A. At each pick, both the source and destination nodes are checked to see if they are at their respective k friend policy limit. If one of them is, the pick is not allowed. This is done until 0 - k picks are found for all users in Ξ.

The Path Traversal User Selection algorithm as shown in Algorithm 2 is a more complex defense scheme that uses a depth first search to create the FSE sub-network. This is done by picking a single top friend of the current user, and in iterations moving through the chosen nodes until no more picks can be made. This algorithm differentiates from Algorithm 1 because it attempts to optimize the first picks of every user in a path rather than just the current user. Checking the k friend limit at each selection and

**Algorithm 2:** FriendGuard Path Traversal User Selection

```
/* N_i is the node selected where c denotes the
   current node. K is the friend policy specified
   by a user.                                      */
Data: ( N_c )
/* net is the main OSN, whereas sub_net is the FSE
   specific sub-network. Calling
   sub_net[N_i].friends.size() gives the current
   connections of N_i.                             */
k = N_c.k_friend_policy();
friends_left = k - sub_net[N_c].friends.size();
for ( N_i ∈ net ) do
    if ( friends_left < k ) then
        if ( net[N_c][N_i] are friends && N_i != N_c &&
          sub_net[N_i].friends.size() < k ) then
            if ( N_i ∈ sub_net[N_c].friends ) then
                continue;
            else
                sub_net[N_c].friends.append[N_i];
                sub_net[N_i].friends.append[N_c];
                friends_left -= 1;
                DFS( N_i, N_i.k_friend_policy() );
            end
        end
    else
        break;
    end
end
```

forcing bidirectional picks is still part of this algorithm to keep the integrity of our core defense.

**Algorithm 3:** FriendGuard Degree Based Path Traversal User Selection

```
/* Sorted is a set containing all nodes of the OSN.
   This list contains nodes in increasing order of
   degree from the original network.               */
Data: (Sorted)
/* N_i denotes the node selected. Calling
   sub_net[N_i].friends.size() gives the current
   connections of N_i.                             */
for ( N_i ∈ Sorted ) do
    if ( sub_net[N_i].friends.size() < k ) then
        Algorithm 2( N_i, N_i.k_friend_policy() );
    else
        continue;
    end
end
```

The Degree Based Path Traversal User Selection algorithm (Algorithm 3) is an optimized depth first search algorithm. In this method, a sort method is conducted prior so that the nodes with lowest degree are picked first. This then utilizes Algorithm 2 to complete the sub-network. Utilizing this method helps to allow low degree nodes to make connections before all their connections within the online social network are selected. The resulting sub-network with this method would hopefully have a reduced number of nodes that return an empty set due to a lack of connections. While other algorithms are able to make connections at runtime by the FSE, the full sub-network would need to be made in this case so that low degree nodes can be made first.

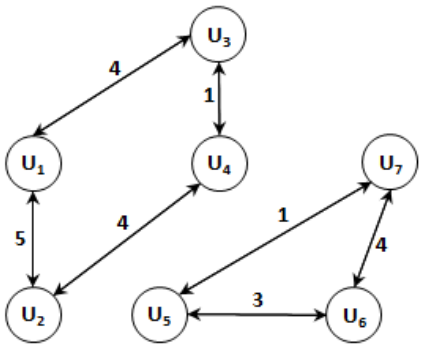



**Figure 4: Unweighted FSE Sub-Network**

Among the above three algorithms, Algorithm 1 and 2 have a nice feature that the construction of the friend search sub-network can be carried out incrementally. In other words, the two algorithms only need to be called during the first query of each node to create the connections that will be displayed to the user at each later query.

Recall the example in Figure 3, the corresponding friend search sub-network obtained by our unweighted algorithm is shown in Figure 4.

## 4.2 Popularity-Based Friend Selection

The popularity-based friend selection aims to return a user's popular friends, i.e., active users on OSN, since active users may have more potential to help increase social activities when recommended to new users. It is nearly impossible to provide $k$ top friends for each user while preserving the security of our system. Therefore, we employ the following trade off during the friend search. If a connection with the most popular friend in the extracted sub-network will leak more than $k$ contacts of an individual user, we will substitute this connection with a less popular connection if possible.

Algorithms 1 and 2 can be modified to utilize weights rather than the node order. Adding weights to these algorithms only slightly changes their overall design. As shown in Algorithms 4 and 5, the weighted algorithm adds another step to check each friend of the current node. This is done in two different ways: a prioritization of the current node's optimal friends, or a compromise between the current and destination node's optimal picks. The two options may lead to similar results in many cases, but there could be substantial differences in some cases. Since all the connections within the FSE sub-network must be bidirectional at the end to prevent additional information leakage, considering both friends' optimal picks may be a more fair choice when the selection process occurs. Figure

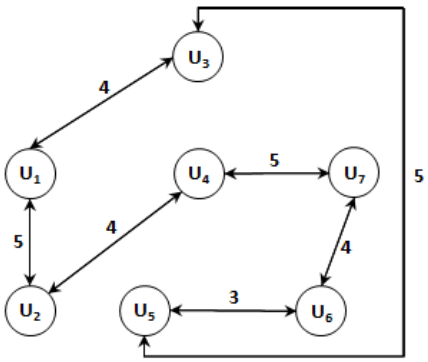



**Figure 5: Weighted FSE Sub-Network**

5 shows the resulting sub-network from Figure 3 when utilizing popularity-based (or weighted) selections.

### 4.3 Trade-off between Privacy and Usability

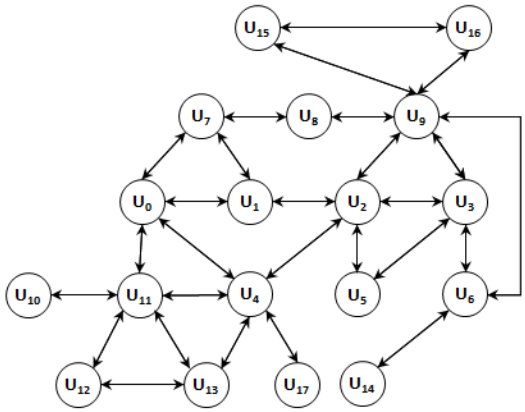



**Figure 6: Sample OSN**

Figure 6 shows another example OSN. The FSE sub-network extracted by our unweighted friend search algorithm is shown in Figure 8. It is assumed that all users have picked k = 2 within this OSN for simple illustration. This example illustrates a trade off to be made between privacy and usability. The impact of this will also be evaluated and discussed in Section 5.1.

The FriendGuard Path Traversal User Selection algorithm was used here to make selections. Figure 7 shows the picks made by this algorithm. Each circled number indicates a root node that has been chosen. Then, the algorithm follows the sub-network to find other nodes connected to the root nodes until not any new node can be added. The nodes with a $*$ next to them denote a connection already made by a previous bidirectional pick. A ! denotes that a node was unable to find any other new connected node. Arrows in the figure represent that the selection algorithm finds another path starting from the root node. This algorithm continues until

**Algorithm 4:** Directional Weight Selection

```
/* Ξ is a set of nodes within an OSN. This will be
   a single node for DFS versions of this, and one
   to the total amount of nodes for the Greedy
   algorithm.                                        */
Data: (Ξ)
/* net is the main OSN, whereas sub_net is the FSE
   specific sub-network. Calling
   sub_net[N_i].friends.size() gives the current
   connections of N_i.                               */
for ( N_i ∈ Ξ ) do
    k = N_i.k_friend_policy();
    friends_left = k - sub_net[N_i].friends.size();
    while ( friends_left < k ) do
        highest_weight = 0;
        N_h = -1;
        for ( N_j ∈ net ) do
            if ( net[N_i][N_j] are friends && N_i != N_j &&
              sub_net[N_j].friends.size() < k ) then
                if ( net[N_i][N_j].weight() > highest_weight &&
                  N_j ∉ sub_net[N_i].friends ) then
                    highest_weight = net[N_i][N_j].weight() +
                      network[N_j][N_i].weight();
                    N_h = N_j;
                end
            end
        end
        if ( N_j == −1 ) then
            break;
        else
            sub_net[N_i].friends.append[N_j];
            sub_net[N_j].friends.append[N_i];
            friends_left -= 1;
            if ( Algorithm_Type == DFS ) then
                DFS( N_j, N_j.k_friend_policy() );
            end
        end
    end
end
```

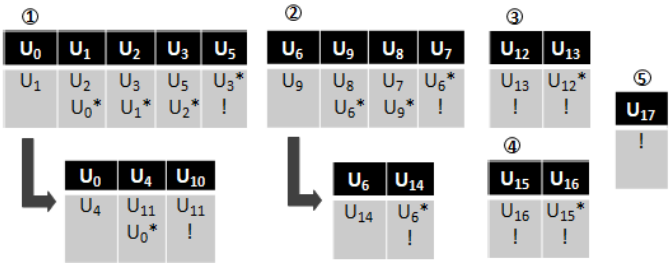



**Figure 7: FriendGuard Path Traversal User Selection**

**Algorithm 5:** Bidirectional Weight Selection

```
Data: ( Ξ )
/* Algorithm 5 follows all the algorithm of
   Algorithm 4 besides the selection of the
   highest node within the friend connections.
   This is done through the variation of this
   selection line within Algorithm 5. All other
   lines are exactly the same before and after
   this line denoted by ...;                    */
...;
if ( net[N_i][N_j] are friends && N_i != N_j &&
 sub_net[N_j].friends.size() < k ) then
    N_h = N_j;
    if ( ( net[N_i][N_j].weight() + net[N_j][N_i].weight() ) >
     highest_weight && N_j ∉ sub_net[N_i].friends ) then
        highest_weight = net[N_i][N_j].weight() +
          network[N_j][N_i].weight();
        N_h = N_j;
    end
end
...;
```

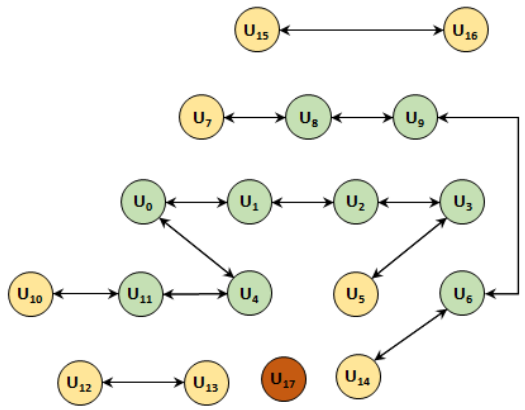



**Figure 8: FSE Sub-Network**

all nodes have reached their $k$ connections or have had a chance to make a selection.

Figure 8 is the resulting friend search sub-network from the original sample osn. Nodes highlighted in green have maximum k connections of 2, yellow represents having a (0, k) connections of 1, and orange represents a k connection of 0. In this case, users may have differing connection degrees in the FSE sub-network even with a constant k friend policy. In a network that has limited connections or an unoptimized path, some nodes of the network may be unused or have less than k connections to make sure there is no more than the max allotted connections for any user. This trade off allows for a more secure sub-network for the FSE, but limits the degree of some users in the sub-network. While this does limit some FSE queries, these loses will be mitigated by the fact that an OSN is usually heavily connected and that some people will opt out of displaying any friends. This will in turn free up some space for less connected nodes.

## 5 EXPERIMENTAL STUDY

Our FriendGuard scheme was tested using three unweighted datasets from Facebook, Twitter, and Google+ [10], along with one Bitcoin weighted dataset [3, 4]. We tested efficiency of our algorithms on the unweighted data, whereas we used the BitCoin dataset to assess algorithm effectiveness. Effectiveness is defined as the most overall optimized sub-network, and measured by total weight of all connections in the sub-network for the FSE. Testing was done by creating fully populated sub-networks.

These datasets were extracted into two-dimensional vectors [3, 4, 10]. Table 3 shows the size of each network. To ensure fair experimental conditions, we kept the same k-friend policy for each user, while in an actual OSN users can pick different policies based on their own privacy preferences.

**Table 3: Social Network Datasets**

| | Facebook | Twitter | Google+ | Bitcoin |
|---|---|---|---|---|
| Nodes | 4,039 | 81,306 | 107,614 | 5,881 |
| Edges | 88,234 | 1,768,149 | 13,673,453 | 35,592 |
| Weighted | False | False | False | True |

### 5.1 Sub-Network Node Degree

In the first set of experiments, we evaluate the effectiveness of our algorithms by varying $k$ from 1 to 20. Then, we examine the percentage of nodes of different degrees in the obtained sub-network. Figure 9 shows the results from the Path Traversal User Selection algorithm. Observe that the percentage of nodes of Degree $k$ decreases with the increase of $k$. This is expected since the larger the value of $k$, (i.e.each user needs to disclose more friends), the

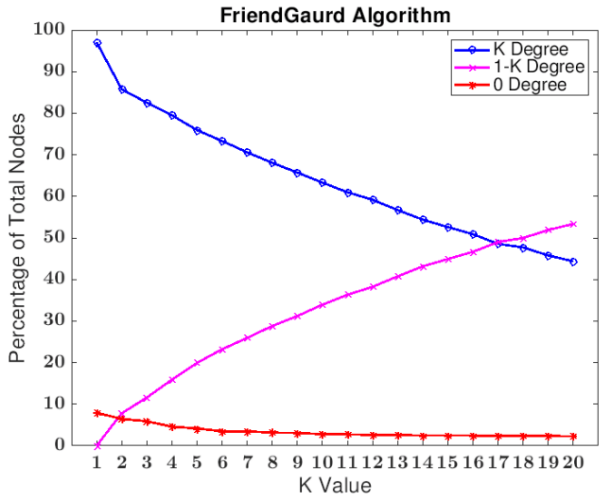
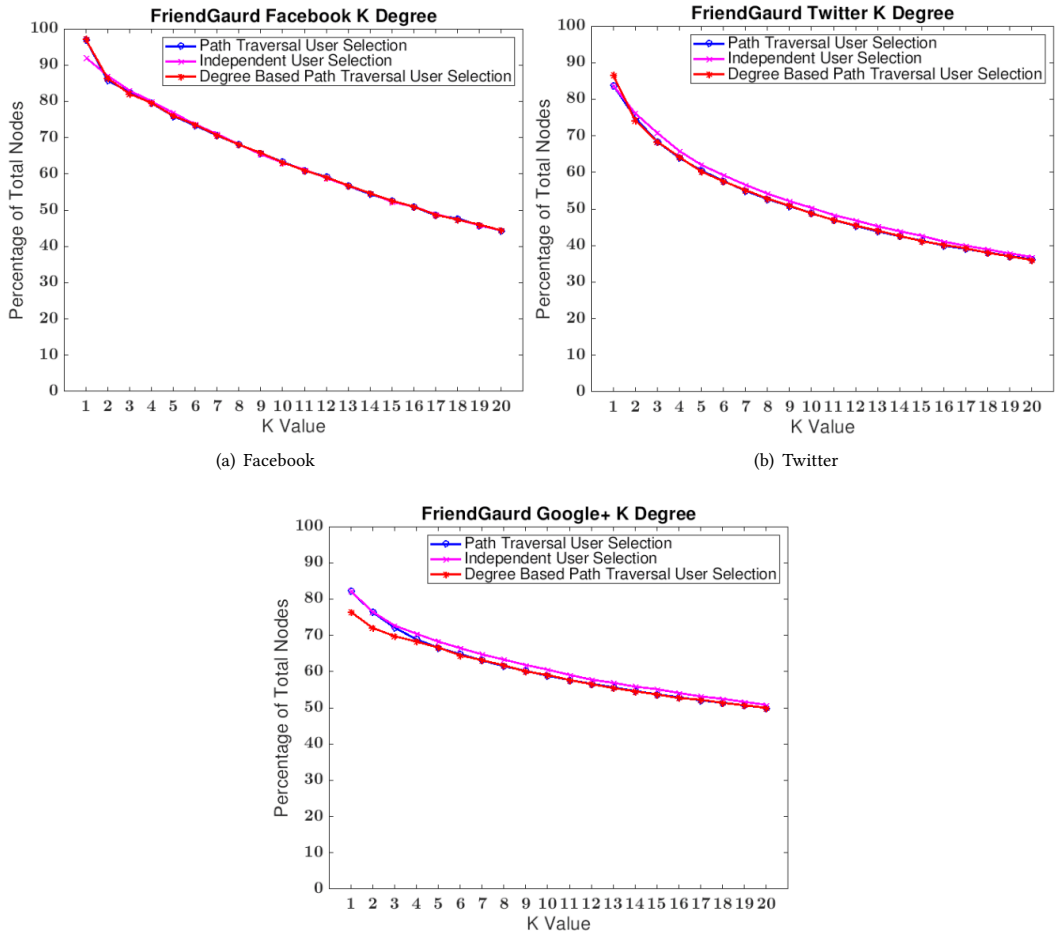



**Figure 9: Perfomance of Path Traversal User Selection Algorithm in Facebook Dataset**




(a) Facebook

(b) Twitter

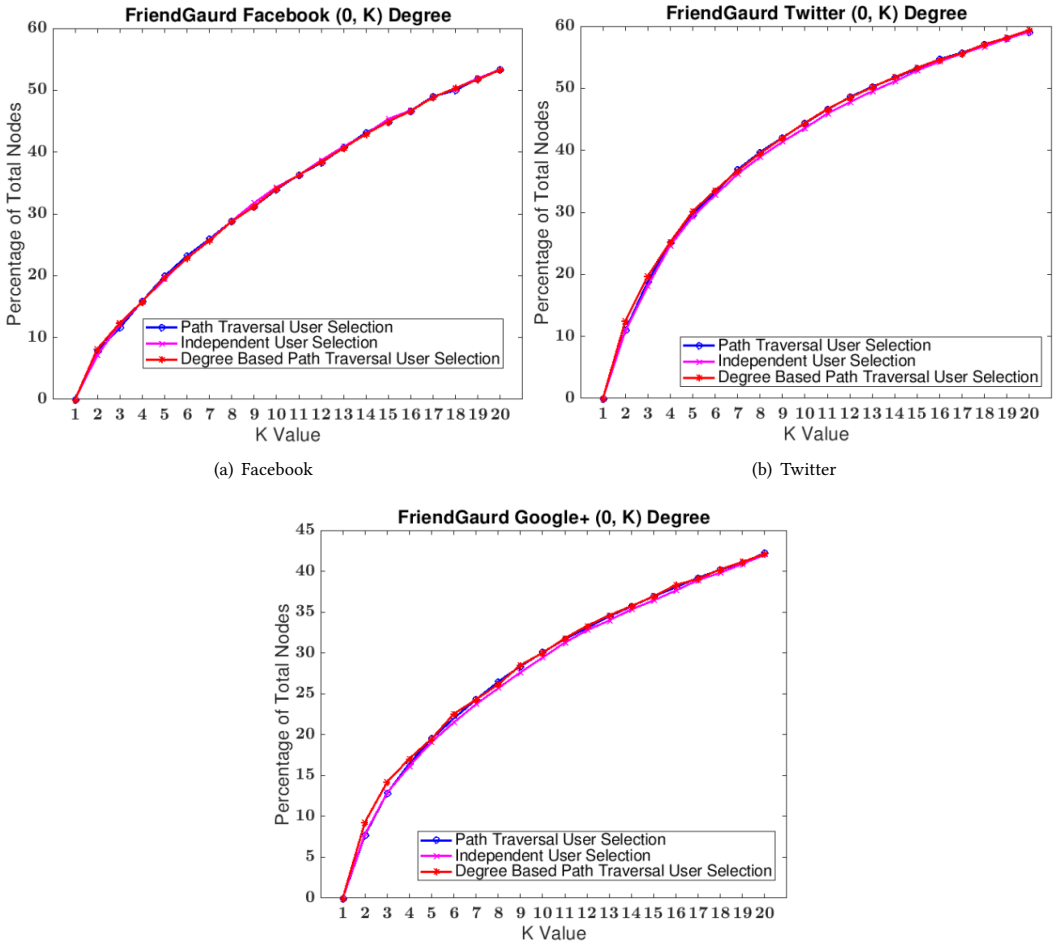
(c) Google+

**Figure 10: Percentage of Nodes with Degree $k$**

harder to find $k$ mutual friends in the network that will not cause additional friend information disclosure. Although the extracted sub-network will not return $k$ friends for some queries, we can see that the sub-network still has a high percentage of nodes with degrees ranging between 1 to $k$, and very low percentage (less than 5%) of nodes with degree 0 (i.e., no friend to report) in most cases, which means the majority of friend queries on the sub-network will return non-empty results.

Our two other algorithms, the Degree Based Path Traversal User Selection and the Independent User Selection algorithm demonstrate similar effectiveness as shown in Figures 10, 11 and 12. Recall that the Path Traversal User Selection algorithm attempts to optimize first picks for each user. The Degree Based Path Traversal User Selection that employs the same strategy but picks low degree nodes first, and the Independent User Selection algorithm aims to optimize all of a single user's picks. In the experiments, the Degree Based Path Traversal User Selection algorithm achieved slightly higher $k$ percentages noted in Figure 10, and slightly lower 0 percentage nodes in Figure 12 than the regular Path Traversal User Selection algorithm for all datasets besides Google+. This is expected because the Degree Based Path Traversal User Selection prioritizes nodes with lower degree. This helps these nodes to get picks before their connections have $k$ connections and are unable to be picked. The Independent User Selection algorithm preformed slightly better than the Path Traversal User Selection algorithm in all cases. It preformed slightly better or around the same for the Degree Based Path Traversal User Selection algorithm as $k$ grew, and a littler worse when $k$ was low. While these algorithms have different methods and overall performance. They all have very similar node degree results as $k$ grows.

As $k$ increases, the FSE sub-network moves from having mostly $k$ degree nodes to mostly $(0, k)$ degree nodes. This could be due to the average degree of nodes being less than the corresponding $k$ values. While this would not be an optimal return set, it still gives most of the desired functionality of the FSE for users. With no nodes having higher than $k$ degree, our approach gives most users full or close to full $k$ sets from the FSE without any vulnerabilities.




(a) Facebook

(b) Twitter

(c) Google+

**Figure 11: Percentage of Nodes with Degrees Between 0 and $k$**

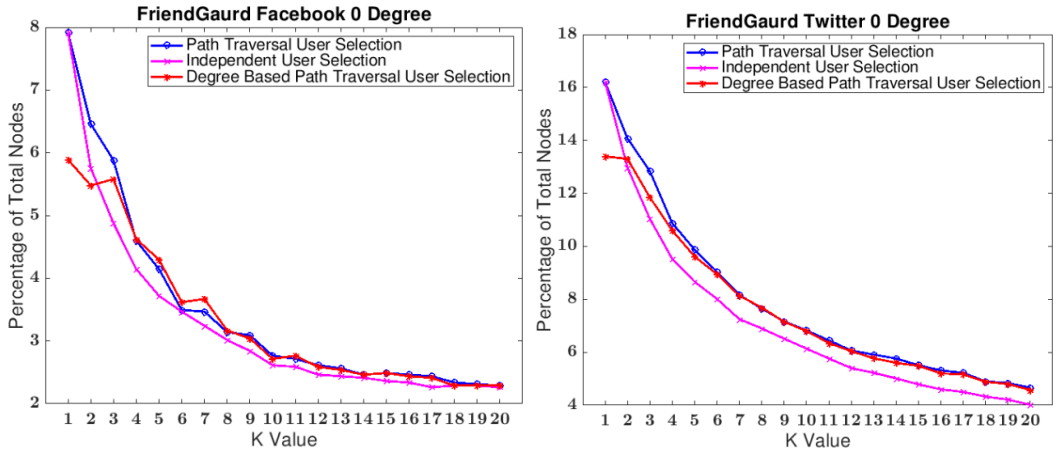
Since our system, in theory, may cause some nodes to have no friends to report, we hereby evaluate its effect in the real datasets. With small values of $k$, the percentage of unusable nodes having 0 connections is between 5 - 24 % in Figure 12. This is not the majority of nodes, but still heavily reduces the functionality of the FSE. Because our testing method keeps $k$ constant for all users, having low $k$ values heavily affects nodes that have a low degree in the original OSN. This drawback is substantially reduced as $k$ increases. Across all datasets, the percent of nodes with 0 connections is between 2 - 9 % for $k$ = 20. This could be perceived as the expected number of users who have a $k$ friend policy of 0. While these are average results across three different datasets, the percentage of nodes in the FSE sub-networks will heavily depend on the average degree of nodes, and the adopted $k$ friend policies for each user. It is likely that most results will be similar or better as OSNs are usually highly connective. Our average degree per node in the Facebook dataset is around 22 friends whereas the real average is 338 friends [13]. This should help to increase the functionality of our defense scheme when utilized in a very large OSN.

## 5.2 Node Connections Optimized by Weight

The Bitcoin dataset was used to measure the aggregated weight of all connections in the FSE sub-network. This test was preformed using the two methods shown by Algorithms 4 and 5, respectively. Tests were done for both the Path Traversal User Selection and Independent User Selection versions of the algorithms, varying 1-20 $k$ policies. As in the prior experiment, the policies were kept constant for every node.

Shown in Figure 13 is the aggregated weight for both the tested algorithms with two different choices. Directional Weight Selection was done by picking weights in deceasing order of connections from source to destination node. Bidirectional Weight Selection was also picked in decreasing order, but picked by sum of weights from source to destination and destination to source.

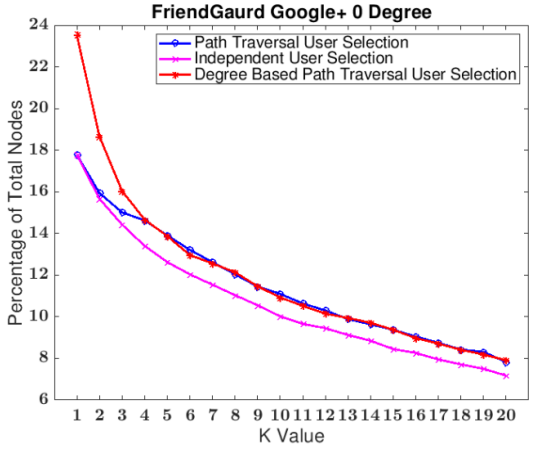
We find these results not surprising. Due to prior testing done in 5.1, it was expected that weights for both Path Traversal User Selection and Independent User Selection algorithms would be similar due to their near identical node degrees across the datasets. The Independent User Selection algorithm did have a slightly lower




(a) Facebook




(b) Twitter




(c) Google+

**Figure 12: Percentage of Nodes with Degree 0**

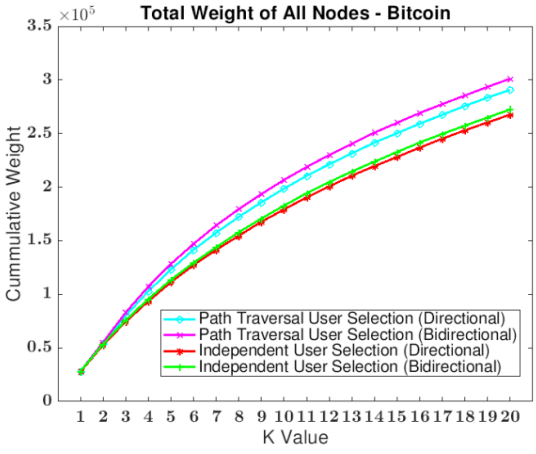



**Figure 13: Effect of Weighted Nodes**

total weight across all nodes. This could be due to the fact that it selects highest weight nodes for one user at a time, whereas the Path Traversal User Selection algorithm selects a single highest degree node at each iteration in the path from the starting node. This could allow for more nodes to get their highest pick which would result in a higher net weight if each method has a similar percentage of degree among the nodes. Bidirectional weight selection was consistently higher than than directional weight selection. This is again due to the fact that this method optimizes picks based on both the source and destination weights. Because node selections must be bidirectional, a directional weight scheme can lead to a situation where one node has a high weight and its selection a low one. This will usually result in a lower total weight than a compromised selection.

Both weight selection techniques have very similar runtimes compared to one another. In this experiment, bidirectional weight selection makes consistently higher weighted picks in this test network. While the Independent User Selection algorithm performs

slightly worse by comparison with the Path Traversal User Selection method, either algorithm can be run with similar results. This test emphasizes that our method can be run under various settings, produce similar results and still be able to maintain a strong defense against information overexposure.

## 5.3 Full Sub-Network Runtime

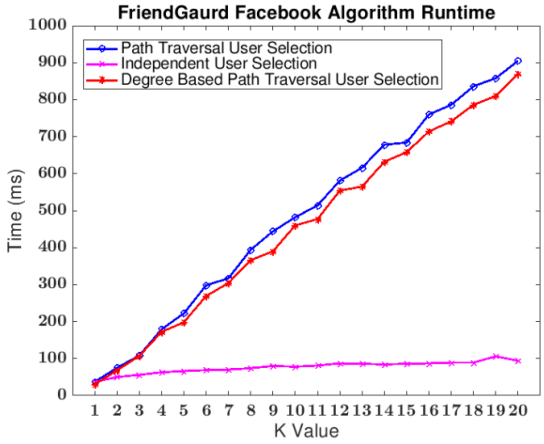



**Figure 14: Facebook FSE Runtime**

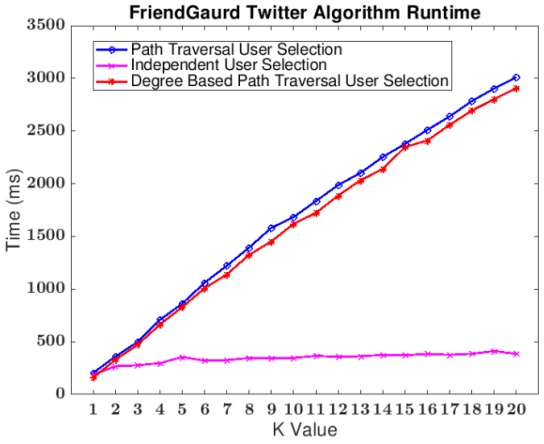



**Figure 15: Twitter FSE Runtime**

Lastly, all unweighted datasets were tested to measure time performance. In this test, Twitter and Google+ were capped at 8000 node networks. Results are reported in Figure 16, Figure 14 and Figure 15. The Independent User Selection algorithm had a much lower runtime than the both the Path Traversal User Selection, and Degree Based Path Traversal User Selection algorithms. While both Path Traversal User Selection algorithms perform similarly, it must be noted that our reported results do not account for the creation of a sorted list needed for the Degree Based Path Traversal User Selection. The sorted list took some time to create putting this algorithm behind the speed of the original Path Traversal User

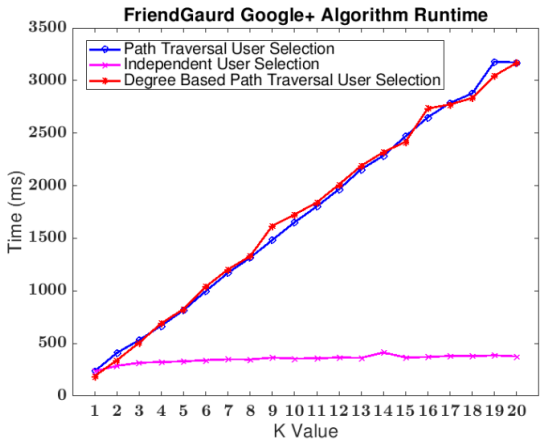



**Figure 16: Google+ FSE Runtime**

Selection when accounting for the creation of a sorted list. While all algorithms show a fairly linear runtime, the sub-network does not need to be created before any nodes are queried by the FSE.

Running each algorithm as a node is queried seems to be the optimal way of making the FSE sub-network. While the Degree Based Path Traversal User Selection algorithm gives slightly better performance in node degree and optimized picks noted in Section 5.1, this algorithm cannot be made at runtime. Our results show that both the Path Traversal User Selection and Independent User Selection algorithm at $k = 8$ (the standard for the original Facebook FSE when vulnerabilities were found [1]) average below 1ms for each node. This would allow our sub-network to be set up in iterations which additionally helps to make connections with more recent data. Due to the low average runtime of each node, this type of setup would have little to no impact on a user making FSE queries.

## 6 DISCUSSION

As security grows, most companies still see optimization of processes as one of their biggest concerns. If a solution can not be optimized in a way where normal actions do not impede the user, it is unlikely the solution will be adopted by the companies. Therefore, our proposed friend search engine strives to achieve high efficiency while adding extra privacy protection. Employing either the Path Traversal User Selection or Independent User Selection algorithm works very quickly when only run on one node. This allows a sub-network to be setup incrementally, enabling quick friend search engine queries, while offering a secure environment in the back end.

It is worth noting that our defense scheme requires updates to the constructed sub-network when someone changes his/her friend policy to a lower $k$ value. This is due to the fact that sets from the friend search engine are given from sub-network connections rather than being created in real time. In this case, parts of the sub-network must be removed so that the security integrity of our approach is kept. This would require a complexity of $O(2kc)$ in the worst case because all connections to the changed node must be removed. This is equally true when the popularity scores of users

are changed during the use of the OSN. While this is a low impact on the performance of our solution, it must be kept in mind that the sub-network will be continuously changing in response to user policy changes and social activities .

The change to the sub-network leads to another possible attack that has not been discussed in the existing friend search engine defense schemes [6, 12]. That is attackers may leverage the historical query results to gain more knowledge. For example, consider user A with a friend policy $k$ of 3. Assume that at the beginning user A's top friends are users B, C, and D, but after one year their top friends change to users E, F, and G. If someone had tracked this information over time, it is likely to say that they now know user A is still connected to users B-D, and is also connected to users E-G. This violates user A's friend policy which only allows the disclosure of 3 of his/her friends. This is true for our defense scheme and possibly the defense schemes in [6] and [12] if common and popular connections change over time. While our defense can mitigate this by never changing the original setup of the friend search engine sub-network, this heavily mitigates the overall relevance of the friend search engine. This is just another consideration to be made when setting up defense schemes, and if security should be consistent for its original design or should be consistent across its lifetime. Due to the nature of the friend search engine though, it is hard to say if any defense scheme can be resistant to this attack without heavily reducing the relevance of the friend search engine, which is the trade off that our defense scheme can make.

## 7 CONCLUSION

In this paper, we have proposed a novel friend search engine called FriendGuard which honors users privacy policies and guarantees friend exposure degree. The FriendGuard system is based on the idea of constructing a reliable sub-network that eliminates vulnerabilities brought by aggregating query results from different rounds to a friend search engine. This sub-network can be created prior to and updated incrementally in response to the changes of social network connections as well as users' social activities. The FriendGuard supports two kinds of friend searches including unweighted and popularity-based friend search, and we have developed several algorithms. In the experiments, we have evaluated and compared our algorithms in different real network datasets. The experimental results demonstrate both efficiency and effectiveness of our approach. It is believed that by using our system, the friend search engine will be a more secure environment at the time of implementation while additionally giving users more flexibility in their privacy controls.

## ACKNOWLEDGEMENT

This work is partially supported by National Science Foundation under the project DGE-1914771.Work from Dr Squicciarini was partially funded by National Science Foundation under grant 1453080.

## REFERENCES

[1] Joseph Bonneau, Jonathan Anderson, Ross Anderson, and Frank Stajano. 2009. Eight friends are enough: Social graph approximation via public listings. *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems* (2009), 13–18.

[2] Safia Bourahla and Yacine Challal. 2017. Social Networks Privacy Preserving Data Publishing. *International Conference on Computational Intelligence and Security (CIS)* (2017), 258–262.

[3] Srijan Kumar, B. Hooi, D. Makhija, M. kumar, V. Subrahamanian, and Christos Faloutsos. 2018. REV2: Fraudulent User Prediction in Rating Platforms. *11th ACM International Conference on Web Search and Data Mining (WSDM)* (2018).

[4] Srijan Kumar, Francesca Spezzano, V. Subrahamanian, and Christos Faloutsos. 2016. Edge Weight Prediction in Weighted Signed Networks. *IEEE 16th International Conference on Data Mining (ICDM)* (2016), 221–230.

[5] Lihui Lan and Lijun Tian. 2013. Preserving Social Network Privacy Using Edge Vector Perturbation. *International Conference on Information Science and Cloud Computing Companion* (2013), 188–193.

[6] Na Li. 2014. Privacy-aware display strategy in friend search. *IEEE International Conference on Communications (ICC)* (2014), 945–950.

[7] Dan Lin, Elisa Bertino, Reynold Cheng, and Sunil Prabhakar. 2009. Location Privacy in Moving-Object Environments. *Trans. Data Privacy* 2, 1 (April 2009), 21–46.

[8] Yuhong Liu and Na Li. 2016. An Advanced Collusion Attack against User Friendship Privacy in OSNs. *IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)* 1 (2016), 465–470.

[9] Yuhong Liu and Na Li. 2019. Retrieving Hidden Friends: A Collusion Privacy Attack Against Online Friend Search Engine. *IEEE Transactions on Information Forensics and Security* 14, 4 (2019), 833–847.

[10] J. McAuley and J. Leskovec. 2012. Learning to Discover Social Circles in Ego Networks. (2012).

[11] Frank Nagle and Lisa Singh. 2009. Can Friends Be Trusted? Exploring Privacy in Online Social Networks. *International Conference on Advances in Social Network Analysis and Mining* (2009), 312–315.

[12] Wei Ren, Shiyong Huang, Yi Ren, and Choo Kim-Kwang. 2016. LiPISC: A Lightweight and Flexible Method for Privacy-Aware Intersection Set Computation. https://doi.org/10.1371/journal.pone.0157752. (2016).

[13] Aaron Smith. 2014. What people like and dislike about Facebook. (Feb 2014). Retrieved February 2, 2019 from http://www.pewresearch.org/fact-tank/2014/02/03/what-people-like-dislike-about-facebook/

[14] Anna Squicciarini, Sushama Karumanchi, Dan Lin, and Nicole Desisto. 2014. Identifying Hidden Social Circles for Advanced Privacy Configuration. *Computer and Security* 41 (2014), 40–51.

[15] A. Squicciarini, D. Lin, S. Karumanchi, and N. DeSisto. 2012. Automatic social group organization and privacy management. In *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*. 89–96.

[16] A. C. Squicciarini, D. Lin, S. Sundareswaran, and J. Wede. 2015. Privacy Policy Inference of User-Uploaded Images on Content Sharing Sites. *IEEE Transactions on Knowledge and Data Engineering* 27, 1 (2015), 193–206.

[17] Anna Cinzia Squicciarini, Smitha Sundareswaran, Dan Lin, and Joshua Wede. 2011. A3P: adaptive policy prediction for shared images over popular content sharing sites. In *HT'11, Proceedings of the 22nd ACM Conference on Hypertext and Hypermedia, Eindhoven, The Netherlands, June 6-9, 2011*. 261–270.

[18] Aira Yamada, Tiffany Kim, and Adrian Perrig. 2012. Exploiting privacy policy conflicts in online social networks. (2012), 1–9.