A Two-stage Deanonymization Attack Against Anonymized Social Networks

Wei Peng, *Student Member, IEEE,* {Feng Li, Xukai Zou}, *Member, IEEE,* and Jie Wu, *Fellow, IEEE*

Abstract—Digital traces left by users of online social networking services, even after anonymization, are susceptible to privacy breaches. This is exacerbated by the increasing overlap in user-bases among various services. To alert fellow researchers in both the academia and the industry to the feasibility of such an attack, we propose an algorithm, *Seed-and-Grow*, to identify users from an anonymized social graph, based solely on graph structure. The algorithm first identifies a *seed* sub-graph, either planted by an attacker or divulged by a collusion of a small group of users, and then *grows* the seed larger based on the attacker's existing knowledge of the users' social relations. Our work identifies and relaxes implicit assumptions taken by previous works, eliminates arbitrary parameters, and improves identification effectiveness and accuracy. Simulations on real-world collected datasets verify our claim.

Index Terms—social networks, anonymity, privacy, attack, graph

1 Introduction

Internet-based social networking services are prevalent in modern societies: a lunch-time walk across a university campus in the United States provides enough evidence. As Alexa's Top 500 Global Sites statistics (retrieved on May 2011) indicate, Facebook and Twitter, two popular online social networking services, rank at 2nd and 9th place, respectively.

One characteristic of online social networking services is their emphasis on the users and their connections, in addition to the content as seen in traditional Internet services. Online social networking services, while providing convenience to users, accumulate a treasure of user-generated content and users' social connections, which were only available to large telecommunication service providers and intelligence agencies a decade ago.

Online social networking data, once published, are of great interest to a large audience: Sociologists can verify hypotheses on social structures and human behavior patterns; third-party application developers can produce value-added services such as games based on users' contact lists; advertisers can more accurately infer a user's demographic and preference profile and can thus issue targeted advertisements. As the December 2010 revision of Facebook's Privacy Policy phrases it: "We allow advertisers to choose the characteristics of users who will see their advertisements and we may use any of the non-personally identifiable attributes we have collected (including information you may have decided not to show to other users, such as your birth year or other sensitive personal information or preferences) to select the appropriate audience for those advertisements."

- *W. Peng and Dr. X. Zou are with the Department of Computer and Information Science, Indiana University-Purdue University Indianapolis, Indianapolis, IN, 46202.*
- *Dr. F. Li is with the Department of Computer, Information, and Technology, Indiana University-Purdue University Indianapolis, Indianapolis, IN, 46202.*
- *Dr. J. Wu is with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA, 19122.*

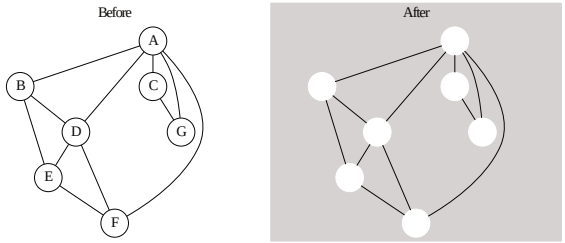


Fig. 1. Naive anonymization removes the ID, but retains the network structure.

Due to the strong correlation to users' social identity, privacy is a major concern in dealing with social network data in contexts such as storage, processing and publishing. Privacy control, through which users can tune the visibility of their profile, is an essential feature in any major social networking service [1].

A common practice in publishing social network is anonymization, i.e., removing plainly identifying labels such as names, social security numbers, postal or e-mail addresses, but retaining the network structure. Figure 1 illustrates this process. The motivation behind such anonymization is that, by removing the "who" information, the utility of the social networks is maximally preserved without compromising users' privacy. In several high-profile cases, anonymity has been unquestioningly interpreted as equivalent to privacy [2].

Can the aforementioned "naive" anonymization technique achieve privacy preservation in the context of privacy-sensitive social network data publishing? This interesting and important question was posed only recently [3]. A few privacy attacks have been proposed to circumvent the naive anonymization protection [2, 3]. Meanwhile, more sophisticated anonymization tech-

niques have been proposed to provide better privacy protection [4, 5, 6, 7, 8]. Nevertheless, research in this area is still in its infancy and a lot of work, both in attacks and defenses, remains to be done.

In this paper, we describe a two-stage identification attack, *Seed-and-Grow*, against anonymized social networks. The name suggests a metaphor for visualizing its structure and procedure. The attacker first plants a *seed* into the target social network before its release. After the anonymized data is published, the attacker retrieves the seed and makes it *grow* larger, thereby further breaching privacy.

More concretely, our contributions include:

- We propose an efficient seed construction and recovery algorithm (Section 3.1). More specifically, we drop the assumption that the attacker has complete control over the connection between the seed and the rest of the graph (Section 3.1.2); the seed is constructed in a way which is only visible to the attacker (Section 3.1.2); the seed recovery algorithm examines at most the two-hop local neighborhood of each node, and thus is efficient (Section 3.1.3).
- We propose an algorithm which grows the seed (i.e., further identifies users and hence violates their privacy) by exploiting the overlapping user bases among social network services. Unlike previous works which require arbitrary parameters for probing aggressiveness, our algorithm automatically finds a good balance between identification effectiveness and accuracy (Section 3.2).
- We demonstrate the significant improvements in identification effectiveness and accuracy of our algorithm over previous works with real-world social-network datasets (Section 4).

2 Background and Related Work

A natural mathematical model to represent a social network is a graph. A graph G consists of a set V of vertices and a set $E \subseteq V \times V$ of edges. Labels can be attached to both vertices and edges to represent attributes.

In this context, *privacy* can be modeled as the knowledge of existence or absence of vertices, edges, or labels. An extension is to model privacy in terms of metrics, such as betweenness, closeness, and centrality, which originate from *social network analysis* studies [9].

The naive anonymization is to remove those labels which can be uniquely associated with one vertex (or a small group of vertices) from V. This is closely related to traditional anonymization techniques employed on relational datasets [10]. However, the information conveyed in edges and its associated labels is susceptible to privacy breaches. Backstrom et al. [3] proposed an identification attack against anonymized graph, and coined the term *structural steganography*.

Besides privacy, other dimensions in formulating privacy attacks against anonymized social networks, as identified in numerous previous works [5, 6, 8, 11], are the published data's *utility*, and the attacker's *background knowledge*.

Utility of published data measures information loss and distortion in the anonymization process. The more information that is lost or distorted, the less useful published data is. Existing anonymization schemes [4, 5, 6, 8, 11] are all based on the trade-off between the usefulness of the published data and the strength of protection. For example, Hay et al. [8] propose an anonymization algorithm in which the original social graph is partitioned into groups before publication, and "the number of nodes in each partition, along with the density of edges that exist within and across partitions," are published.

Although a trade-off between utility and privacy is necessary, it is hard, if not impossible, to find a proper balance overall. Besides, it is hard to prevent attackers from proactively collecting intelligence on the social network. It is especially relevant today as major online social networking services provide APIs to facilitate third-party application development. These programming interfaces can be abused by a malicious party to gather information about the network.

Background knowledge characterizes the information in the attacker's possession which can be used to compromise privacy protection. It is closely related to what is perceived as privacy in a particular context.

The attacker's background knowledge is not restricted to the target's neighborhood in a single network, but may span multiple networks and include the target's alter egos in all of these networks [2]. This is a realistic assumption. Consider the status quo in the social networking service business, in which service providers, like Facebook and Flickr, offer complementary services. It is very likely that a user of one service would simultaneously use another service. As a person registers to different social networking services, her connections in these services, which relate to her social relationships in the real world, might reveal valuable information which the attacker can make use of to threaten her privacy.

The above observation inspires Seed-and-Grow, which exploits the increasingly overlapping user-bases among social networking services. A concrete example is helpful in understanding this idea.

> [Motivating scenario.] Bob, as an employee of a social networking service provider F-net, acquires from his employer a social-network data-set, in which vertices represent users and edges represent private chat sessions. The edges are labeled with attributes such as timestamps. In accordance with its privacy policy, F-net has removed the user IDs from the graph before giving it to Bob.
> Bob, being an inquisitive person, wants to know who these users are. Suppose, somehow, Bob identifies 4 of these users from the graph (the "Seed Construction" and "Seed Recovery" interludes in Section 3.1 illustrate a way to do this). By using a graph (with the user ID tagged) he crawled a month ago from the website of another service provider T-net (the 4 identified persons are also users of T-net), and by carefully measuring structural similarity of

these graphs, he manages to identify 100 more users from the anonymized graph (the "Dissimilarity" interlude in Section 3.2 illustrates a way to do this). By doing so, Bob circumvents his employer's attempt to protect its customers' privacy.

We conclude this section with a brief comment on our choice of model. We use the *undirected* graph model to explain the proposed deanonymization attack on social networks. Undirected graphs arise naturally in scenarios where the social relation under investigation is *mutual*, e.g., friend requests must be confirmed on Facebook. *Directed* graphs, however, are more appropriate in other cases, such as fans following a movie star on Twitter. An undirected graph could be seen as a special case of directed graphs, in which the relationship is reciprocal; Mislove et al. confirmed the relationship reciprocity in a large-scale study on the Flickr online photo-sharing service [12]. As explained in Section 3, the algorithms used in the proposed deanonymization attack do not rely on the fact that the used graphs are undirected; they work on directed graphs the same way. The undirected graph model is only a choice for specificity and ease of presentation.

3 Seed-and-Grow: The Attack

This section describes an attack that identifies users from an anonymized social graph. Let an undirected graph $G_T = \{V_T, E_T\}$ represent the *target* social network after anonymization. We assume that the attacker has an undirected graph $G_B = \{V_B, E_B\}$ which models his *background knowledge* about the social relationships among a group of people, i.e., V_B are labeled with the identities of these people. The motivating scenario demonstrates one way to obtain G_B.

The attack concerned here is to infer the identities of the vertices V_T by considering *structural similarity* between the target graph G_T and the background graph G_B: Nodes that belong to the same users are assumed to have similar connections in G_T and G_B. Although sporadic connections between who would otherwise be strangers may exist in an online social network (and, thus, affect the similarity between G_T and G_B), such links can be removed by, for example, quantifying the strength of these connections [13]; the residual network consists of the stable, strong connections that reflect the users' real-world social relationships, which give rise to the similarity between G_T and G_B. Additionally, auxiliary knowledge about the target graph G_T (such as the source and nature of the graph) may help in choosing a background graph G_B with similar structures.

Thus, the two graphs G_T and G_B are syntactically (the social connections) similar but semantically (the meaning associated with such connections) different. By re-identifying the vertices in G_T with the help of G_B, the attacker associates the sensitive semantics with users on the anonymized G_T and, thus, compromise the privacy of such users. An example of sensitive semantics is the private chat sessions, and their associated timestamps, in the motivating scenario.

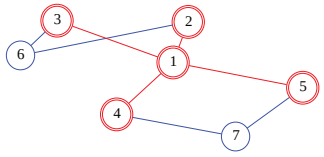


Fig. 2. A randomly generated graph G_F may be symmetric.

We assume that, *before* the release of G_T, the attacker obtains (either by creating or stealing) a few accounts and connects them with a few other users (the *initial seeds*) in G_T. The feasibility of doing this is the basis of the Sybil identity forgery attack studied in numerous previous works [14, 15, 16, 17, 18, 19, 20, 21, 22]. Indeed, experiments (Section 4) show that our algorithm is capable of identifying 10 times of anonymized users from as few as 5 initial seeds. Besides user IDs, the attacker knows nothing about the relationship between the initial seeds and other users in G_T. Furthermore, unlike previous works, we *do not assume that the attacker has complete control over the connections*: the attack only *knows* them before G_T's release. This is more realistic. An example is a confirmation-based social network, in which a connection is established only if the two parties confirm it: the attacker *can decline but not impose* a connection.

In contrast to a pure structure-based vertex matching algorithm [23], Seed-and-Grow is a *two-stage* algorithm.

The *seed* stage plants (by obtaining accounts and establishing relationships) a small specially designed subgraph $G_F = \{V_F, E_F\} \subseteq G_T$ (G_F reads as "fingerprint") into G_T before its release. After the anonymized graph is released, the attacker locates G_F in G_T. The neighboring vertices V_S of G_F in G_T are readily identified and serve as the *initial seeds* to be grown.

The *grow* stage is essentially comprised of a structure-based vertex matching, which further identifies vertices adjacent to the initial seeds V_S. This is a self-reinforcing process, in which the seeds grow larger as more vertices are identified.

3.1 Seed

3.1.1 Feasibility

Successful retrieval of G_F from G_T is guaranteed if G_F exhibits the following structural properties.

- G_F is *uniquely* identifiable, i.e., no subgraph $H \subseteq G_T$ except G_F is isomorphic to G_F. For example, in Figure 2, subgraph $\{v_1, v_2, v_3\}$ is isomorphic to subgraph $\{v_1, v_4, v_5\}$ because there is a structure-preserving mapping $v_1 \mapsto v_1, v_2 \mapsto v_4, v_3 \mapsto v_5$ between them. Therefore, the two subgraphs are structurally indistinguishable once the vertex labels are removed.
- G_F is *asymmetric*, i.e., G_F does not have any non-trivial automorphism. For example, in Figure 2, subgraph $\{v_1, v_2, \ldots, v_5\}$ has an automorphism $v_1 \mapsto v_1, v_2 \mapsto v_3, v_3 \mapsto v_4, v_4 \mapsto v_5, v_5 \mapsto v_2$. Therefore,

even if we could locate $V_F = \{v_1, \ldots, v_5\}$ from G_T, $v_2, \ldots, v_5$ are indistinguishable once their labels are removed.

In practice, since the structure of other nodes in the network is unknown to the attacker before its release, the uniquely identifiable property is not realizable. However, as was proved by Backstrom et al. [3], with a large enough size and randomly generated edges under the Erdös-Rényi model [24], G_F will be uniquely identifiable with high probability.

Although a randomly generated graph G_F is very likely to be uniquely identifiable in G_T, it may violate the asymmetric structural property. However, because the goal of seed is to identify the initial seed V_S rather than the fingerprint G_F, the asymmetric requirement for G_F can be relaxed. For $u \in V_S$, let $V_F(u)$ be the vertices in V_F which connect with u ($|V_F(u)| \geq 1$ by the definition of V_S). For each pair of vertices, say u and v, in V_S, as long as $V_F(u)$ and $V_F(v)$ are distinguishable in G_F (e.g., $|V_F(u)| \neq |V_F(v)|$ or the degree sequences are different; more precisely, no automorphism of G_F exists which maps $V_F(u)$ to $V_F(v)$), and once G_F is recovered from G_T, V_S can be identified uniquely. In Figure 2, since $V_F(6) = \{v_2, v_3\}$ and $V_F(7) = \{v_4, v_5\}$ are not distinguishable, vertices v_6 and v_7 can not be identified through G_F.

Based on these observations, we propose the following method of constructing and recovering G_F.

3.1.2 Construction

The construction of G_F starts with a *star* structure. The motivation for the star structure will become clear in Section 3.1.3. We call the vertex at the center of the star the *head* of G_F and denote it by v_h. v_h connects and *only* connects to every other vertex in G_F.

The vertices in $V_F - \{v_h\}$ are connected with some other vertices of the initial seeds V_S in G_T. To ensure the distinguishability of two seeds u and v once the fingerprint G_F is recovered, the attacker can decline those connection requests (from other vertices in G_T) which render $V_F(u) = V_F(v)$. Note that the attacker is not assumed to have full control over the connections: an attacker does not have to impose a connection as long as he can decline it.

After setting up the initial star structure, the attacker establishes other *internal* connections within the fingerprint graph G_F. Two principles dictate this process:

1) No automorphism of G_F should map $V_F(u)$ to $V_F(v)$ for two distinct initial seeds u and v.
2) The constructed G_F should leave no distinctive structural pattern for anyone besides the attacker, but should yet be recoverable.

Principle 1 follows from the discussion in Section 3.1.1: a pair of initial seeds u and v could be unambiguously identified only if no automorphism of G_F maps $V_F(u)$ to $V_F(v)$. Principle 2 apparently presents a dilemma: G_F should mingle with the rest of the target graph G_T, yet be distinctive. In the following discussion, we first justify this principle, and then resolve the dilemma by reconciling the two competing requirements.

The motivation for having G_F mingle with the rest of the target graph G_T is to avoid leaving distinctive structural patterns for defenders. Otherwise, a straightforward defense against the proposed attack would be to locate the fingerprint graph G_F by pattern-matching and to remove it prior to the publication of G_T. An implication is that the construction of G_F should be stochastic rather than deterministic.

Yet, stochastic construction alone is not enough for G_F to blend into G_T. Numerous studies [25, 26, 27, 28, 29, 30, 31] indicate the existence of distinctive structural properties of online social networks as opposed to arbitrary random graphs. In particular, online social graphs consist of a well-connected backbone linking numerous small communities [25]. Within each community, vertices show a local, transitive, triangle-closing connection pattern [29]. The construction of G_F should reflect these properties to blend into G_T.

The cost for the attacker to set up the fingerprint graph G_F is dominated by the number and variety of connections between V_F and the initial seeds V_S. To minimize the cost, the construction of G_F mimics a local community in G_T [25]: after establishing the star structure centering at the head vertex v_h, each pair of vertices in $V_F - \{v_h\}$ connects with a probability of t. The probability t reflects the *transitivity* of a community in G_T, which is the likelihood that, in the same community, two vertices sharing a common neighbor (v_h in G_F) will connect to each other. In reality, the attacker almost always knows some auxiliary information about the target graph G_F, which may include the community transitivity and a reasonable size for a community: The construction of G_F should be adjusted to such information for G_F to blend into G_T.

After connecting pairs of non-head vertices in V_F with a probability of the community transitivity t, the attacker collects the *internal degree* $D_F(v)$, which is number of vertices in V_F that v connects to, for every $v \in V_F - \{v_h\}$ into an *ordered* sequence $\mathcal{S}_D$.

Now, for every $v \in V_S$, v has a corresponding subsequence $\mathcal{S}_D(v)$ of $\mathcal{S}_D$ according to its connectivity with V_F. For example, in Figure 2, v_6 connects to v_2 and v_3 from G_F; since $D_F(v_2) = D_F(v_3) = 1$, $\mathcal{S}_D(v_6) = \langle 1, 1 \rangle$. As long as $\mathcal{S}_D(u) \neq \mathcal{S}_D(v)$ for u and v from V_S, no automorphism of G_F will map $V_F(u)$ to $V_F(v)$. Therefore, the attacker guarantees unambiguous recovery of V_S by ensuring that the randomly connected G_F satisfies this condition. If not, the attacker will simply redo the random connection among $V_F - \{v_h\}$ until it does (which it eventually will, since we assume that $V_F(u) \neq V_F(v)$ for any pair u and v from V_S). Algorithm 1 summarizes this procedure.

[Seed construction.] Bob had created 7 accounts v_h and $v_1, \ldots, v_6$, i.e., V_F. He first connected v_h with $v_1, \ldots, v_6$. After a while, he noticed that users v_7 to v_{10} are connected with $v_1, \ldots, v_6$, i.e., $V_S = \{v_7, \ldots, v_{10}\}$.

Algorithm 1 Seed construction.

```
Create V_F = {v_h, v_1, v_2, ...}.
Given connectivity between V_F and V_S.
Connect v_h with v for all v ∈ V_F − {v_h}.
loop
    for all pairs v_a ≠ v_b in V_F − {v_h} do
        Connect v_a and v_b with a probability of the community transitivity t.
    end for
    for all u ∈ V_S do
        Find S_D(u).
    end for
    if S_D(u) are mutually distinct for all u ∈ V_S then
        return
    end if
end loop
```

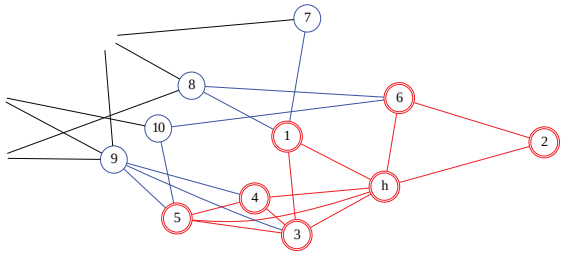


Fig. 3. The task of the seed stage is to identify the initial seed by recovering the fingerprint graph G_F.

Then, he randomly connected $v_1, \ldots, v_6$ with the community transitivity t and got the resulting graph G_F, as shown in Figure 3. The ordered internal degree sequence $\mathcal{S}_D = \langle 2, 2, 2, 3, 3, 4\rangle$.
Bob found out that $\mathcal{S}_D(v_7) = \langle 2\rangle$, $\mathcal{S}_D(v_8) = \langle 2, 2\rangle$, $\mathcal{S}_D(v_9) = \langle 3, 3, 4\rangle$, and $\mathcal{S}_D(v_{10}) = \langle 2, 3\rangle$. Since they are mutually distinct, Bob was sure that he could identify v_7 to v_{10} once V_F was recovered from the published anonymized graph.

The degree of head vertex v_h, the ordered internal degree sequence $\mathcal{S}_D$ and the subsequences chosen for V_S are the *secrets* held by the attacker. These secrets are jointly used to recover G_F from G_T and thereafter to identify V_S (Section 3.1.3). The astronomical combinations of these secrets ensure the high probability that G_F is unambiguously recovered from the anonymized target graph G_T. We present a numerical explanation in Section 4.2. Without knowing the secrets, the attacker could not identify G_F, due to its stochastic construction for blending into G_T. Therefore, G_F is visible only to the attacker, who holds the secrets to G_F.

3.1.3 Recovery

Once G_F has been successfully planted and G_T is released, the recovery of G_F from G_T consists of a systematic check of the attacker's secrets. The first step is to find a candidate u for the head vertex v_h in G_T by degree comparison. Then, the ordered internal degree sequence of the candidate fingerprint graph (i.e., u's 1-hop neighborhood) and the subsequences of the candidate initial seed (i.e., u's 2-hop neighborhood minus its 1-hop neighborhood) are checked against the corresponding secrets. If the candidate fingerprint graph passes these secret checks, it is identified with G_F, and its neighbors are identified with V_S by subsequence secret comparison. Algorithm 2 has the details.

[Seed recovery.] After the anonymized publication of the target graph G_T (with the fingerprint graph G_F planted in it), Bob started to check the vertices in G_T against the secrets of G_F he held. He did this by examining all of the vertices in G_T for one with degree 6. After he had reached a *candidate head* v_c with degree 6, he isolated it along with its immediate neighbors as the candidate fingerprint graph (the red vertices in Figure 3). He found that the ordered internal degree sequence $\langle 2, 2, 2, 3, 3, 4\rangle$ matched that of V_F. He then isolated v_c's 2-hop neighborhood, removed those included in the 1-hop neighborhood, and checked ordered internal degree subsequences of the remaining ones against the secrets. He found that they matched the secrets again.
Bob was now convinced that he had found G_F. By matching the ordered internal degree subsequences of V_c, he identified v_7, v_8, v_9 and v_{10}. For example, for a 2-hop neighbor $u \in V_c$, which connected to three 1-hop neighbors with internal degrees 3, 3 and 4, he identified u with v_9.

The motivation for incorporating the head vertex technique in the seed construction stage becomes clear from the example. The only connections v_h has are *internal* ones. Therefore, once a candidate head vertex u is found, the candidate fingerprint can be readily determined by isolating u's 1-hop neighborhood. No probing or backtracking is needed for finding G_F as in previous works [2, 3].

The efficiency of the algorithm is evident by observing that, in Algorithm 2, the nesting depth of the loops is 3 (2 of them are in a vertex's neighborhood) and no recursion is involved. Because the 2-hop neighborhood of u_v (e.g., $V_F \cup V_S$) is controlled by the attacker (as secrets), if the maximal degree in G_T is N, the complexity of the recovery algorithm is $O(N^2|V_T|)$.

3.2 Grow

The initial seeds V_S provide a firm ground for further identification in the anonymized graph G_T. Background knowledge G_B comes into play at this stage.

We have a partial mapping between G_T and G_B, i.e., the initial seeds V_S in G_T map to corresponding vertices in G_B. Two examples of partial graph mappings are the Twitter and Flickr datasets [2] and the Netflix and IMDB datasets [32]. The straightforward idea of testing all possible mappings for the rest of the vertices has an exponential complexity, which is unacceptable even for a medium-sized network. Besides, the overlapping between G_T and G_B may be *partial*, so a *full* mapping is neither possible nor necessarily desirable. Therefore, the grow algorithm adopts a progressive and self-reinforcing strategy, starting with the initial seeds and extending the mapping to other vertices for each round.

Algorithm 2 Seed recovery.

```
for all u ∈ G_T do
  if deg(u) = |V_F| − 1 then
    U ← exact 1-hop neighborhood of u
    for all v ∈ U do
      d(v) ← number of v's neighbors in U ∪ {u}
    end for
    s(u) ← sort(d(v)|v ∈ U)
    if s(u) = S_D then
      V ← exact 2-hop neighborhood of u
      for all w ∈ V do
        U(w) ← w's neighbors in U
        s(w) ← sort(d(v)|v ∈ U(w))
      end for
      if ⟨s(w)|w ∈ V⟩ = ⟨S_D(v)|v ∈ V_S⟩ then
        {w ∈ V is identified with v ∈ V_S if s(w) = S_D(v)}
      end if
    end if
  end if
end for
```

Figure 4 shows a small example. v_7 to v_{10} have already been identified in the seed stage (recall Figure 3). The task is to identify other vertices in the target graph G_T.

3.2.1 Dissimilarity

At the core of the grow algorithm is a family of related metrics, collectively known as the *dissimilarity* between a pair of vertices from the target and the background graph, respectively. In order to enhance the identification accuracy and to reduce the computation complexity and the false-positive rate, we introduce a *greedy heuristic* with *revisiting* into the algorithm.

It is natural to start with those vertices in G_T which connect to the initial seed V_S because they are more close to the *certain* information, i.e., the already identified vertices V_S. For these vertices, their neighboring vertices can be divided into two groups. Namely, for such a vertex u, its neighborhood in G_T is composed of $\mathcal{N}_m^T(u)$ (*mapped* neighbors) and $\mathcal{N}_u^T(u)$ (*unmapped* neighbors). For instance, in Figure 4, $\mathcal{N}_m^T(u_{*1}) = \{u_7, u_8, u_9\}$ and $\mathcal{N}_u^T(u_{*1}) = \{u_{*4}\}$.

For the background graph G_B, we can make similar definitions. Suppose the seed $V_S \subseteq V_T$ maps to $V_S^* \subseteq V_B$. For a V_S^*'s neighboring vertex v, let $\mathcal{N}_m^B(v)$ be v's neighbors in V_S^*, and let $\mathcal{N}_u^B(v)$ be the other (i.e., unmapped) neighbors. Hence, in Figure 4, $\mathcal{N}_m^B(v_{12}) = \{v_9, v_{10}\}$ and $\mathcal{N}_u^B(v_{12}) = \{v_{11}, v_{16}\}$.

We identify the mapped vertices in V_S and V_S^* so that $\mathcal{N}_m^T(u_{*1}) - \mathcal{N}_m^B(v_{12}) = \{u_7, u_8\} = \{v_7, v_8\}$ in Figure 4.

Our design of the grow algorithm underwent multiple iterations of design-simulation-redesign. The eventual form is deceivingly simple to the point of ad hockery, which apparently makes only partial use of the available information. To appreciate the nuances and illuminate insights, we first present a design along the design iteration which apparently makes fuller use of the available information, but paradoxically is inferior in identification accuracy than the eventual simple design. We show the simulation results confirming this in Section 4.3.4.

TABLE 1
The dissimilarity, as defined by Equations (1)–(4), of the unmapped pairs in Figure 4.

Δ	v_{*1}	v_{*2}	v_{*3}
v_{11}	0.18	0.13	0.54
v_{12}	0.58	0.53	0.09

The complex design. Having categorized vertices in G_T into mapped and unmapped parts, we define the following *dissimilarity* metrics for a pair of *unmapped* vertices $u \in V_T$ and $v \in V_B$, which are connected to the (current and potentially grown) seed V_S and its image V_S^* respectively:

$$\Delta(u, v) = \alpha \Delta_m(u, v) + (1 - \alpha)\Delta_u(u, v), \tag{1}$$

in which

$$\Delta_m(u, v) = \frac{|\mathcal{N}_m^T(u) - \mathcal{N}_m^B(v)| + |\mathcal{N}_m^B(v) - \mathcal{N}_m^T(u)|}{|\mathcal{N}_m^T(u)| + |\mathcal{N}_m^B(v)|}, \tag{2}$$

$$\Delta_u(u, v) = \frac{|\,|\mathcal{N}_u^T(u)| - |\mathcal{N}_u^B(v)|\,|}{\max(|\mathcal{N}_u^T(u)|, |\mathcal{N}_u^B(v)|)}, \tag{3}$$

and[1]

$$\alpha = \frac{1}{2}\left(1 + \frac{\frac{|\mathcal{N}_m^T(u)|}{|\mathcal{N}_m^T(u)|+|\mathcal{N}_u^T(u)|} + \frac{|\mathcal{N}_m^B(v)|}{|\mathcal{N}_m^B(v)|+|\mathcal{N}_u^B(v)|}}{2}\right). \tag{4}$$

$|\cdot|$ is the number of set elements, i.e., the set cardinality. The design follows from the following intuitions.

- The overall dissimilarity ($\Delta(u, v)$) of u and v is a weighted (α) average of dissimilarity for its mapped ($\Delta_m(u, v)$) and unmapped ($\Delta_u(u, v)$) neighborhood. Also, $\Delta(u, v)$ should be symmetric (i.e., $\Delta(u, v) = \Delta(v, u)$). This is because, if we exchange the target and background graphs, the dissimilarity between a specific pair of vertices should be the same.
- $\Delta_m(u, v)$ measures how different u and v's *mapped* neighborhoods are. By its definition in Equation 2, $0 \le \Delta_m(u, v) \le 1$. More precisely, when their mapped neighborhoods are the *same* (i.e., $\mathcal{N}_m^T(u) = \mathcal{N}_m^B(v)$), $\Delta_m(u, v) = 0$, which means u and v match perfectly in regard to their mapped neighborhoods. Otherwise, when $\mathcal{N}_m^T(u) \cap \mathcal{N}_m^B(v) = \emptyset$, $\Delta_m(u, v) = 1$.
- The key difference between the *mapped* and *unmapped* neighborhoods is that the unmapped neighborhoods do not have labels. For instance, in Figure 4, we do not know if the vertices in $\mathcal{N}_u^T(v_{*2}) = \{v_{*3}, \ldots, v_{*7}\}$ and $\mathcal{N}_u^B(v_{11}) = \{v_{12}, \ldots, v_{16}\}$ match. Nevertheless, we can compare them by *degree*. As defined in Equation 3, $0 \le \Delta_u(u, v) \le 1$. More precisely, $\Delta_u(u, v) = 0$ if u and v have the same number of unmapped neighbors; $\Delta_u(u, v) = 1$ when

1. We make a provision for $\Delta_u(u, v) = 0$ if $|\mathcal{N}_u^T(u)| = |\mathcal{N}_u^B(v)| = 0$, i.e., they both have *no* unmapped neighborhoods. There is no need for a similar provision for $\Delta_m(u, v)$ because its denominator, $|\mathcal{N}_m^T(u)| + |\mathcal{N}_m^B(v)|$, is not zero by the definition: u and v both have at least one mapped neighbor.

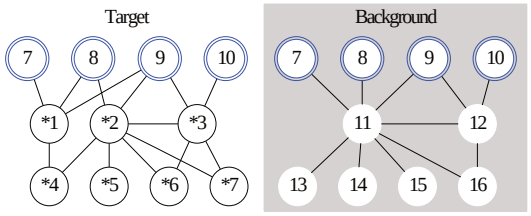


Fig. 4. The task of the grow stage is to identify the unmapped vertices starting from the seed.

one has no unmapped neighbors while the other has.

- The mapped neighborhood provides more *certain* information than the unmapped neighborhood. Hence, the weight for the mapped neighborhood α should be greater. As defined in Equation 4, $\frac{1}{2} \leq \alpha \leq 1$. More precisely, when mapped neighborhoods dominate in both G_T and G_B, $\alpha \to 1$. On the other hand, if unmapped neighborhoods dominate, $\alpha \to \frac{1}{2}$ (The choice of u and v ensures the non-emptiness of mapped neighborhoods).

Table 1 shows a numerical example of the dissimilarity metric for the unmapped vertices in Figure 4. For example, $\Delta_m(v_{*1}, v_{11}) = 0$, $\Delta_u(v_{*1}, v_{11}) = 0.80, \alpha \approx 0.78$. Hence $\Delta(v_{*1}, v_{11}) = 0.78 \times 0 + (1 - 0.78) \times 0.80 \approx 0.18$.

The simple design. At first, we expected that the design presented above, which synthesized information from both mapped and unmapped neighbors, would be accurate in identifying anonymized vertices in G_T; however, simulation results show otherwise (Section 4.3.4). A closer examination indicates two reasons:

- Unmapped neighborhoods are too uncertain to be used in computing dissimilarity, especially through degree as in Equation (3). Two otherwise unrelated vertices may have unmapped neighborhoods with similar degrees and vice versa. Even the biased weight in Equation (4) could not mitigate the negative impact on performance.
- The "single dissimilarity metric" design in Equation (1) conceals a piece of information which is useful for mapping vertices of G_T and G_B in some ambiguous scenarios. Namely, a pair of vertices from G_T and G_B respectively is a good match only if they are *mutually* the best match for one another. There could be multiple best matches for one direction; a mutual best match is a stronger indication for a correct identification.

Based on these insights, we present our improved, yet simple design. For a pair of nodes, $u \in V_T$ and $v \in V_B$, we define the following pair of dissimilarity metrics:

$$\Delta_T(u, v) = \frac{|\mathcal{N}_m^T(u) - \mathcal{N}_m^B(v)|}{|\mathcal{N}_m^T(u)|}, \tag{5}$$

and:

$$\Delta_B(u, v) = \frac{|\mathcal{N}_m^B(v) - \mathcal{N}_m^T(u)|}{|\mathcal{N}_m^B(v)|}, \tag{6}$$

TABLE 2
The dissimilarity, as defined by Equations (5) and (6), of the unmapped pairs in Figure 4. Each tuple consists of a (Δ_T, Δ_B) pair.

Δ	u_{*1}	u_{*2}	u_{*3}
v_{11}	(0.00, 0.00)	(0.00, 0.33)	(0.50, 0.67)
v_{12}	(0.67, 0.50)	(0.50, 0.50)	(0.00, 0.00)

For example, $\Delta_T(u_{*1}, v_{12}) = |\{u_7, u_8\}|/|\{u_7, u_8, u_9\}| = 2/3 \approx 0.667$, and $\Delta_B(u_{*1}, v_{12}) = |\{v_{10}\}|/|\{v_9, v_{10}\}| = 1/2 = 0.5$.

$\Delta_T(u, v)$ and $\Delta_B(u, v)$ together measure how different u and v's mapped neighborhoods are. By its definition in Equations 5 and 6, both $\Delta_T(u, v)$ and $\Delta_B(u, v)$ are in the range of $[0, 1]$. More precisely, when their mapped neighborhoods are the same ($\mathcal{N}_m^T(u) = \mathcal{N}_m^B(v)$), we have $\Delta_T(u, v) = \Delta_B(u, v) = 0$, which means that u and v match perfectly in regard to their mapped neighborhoods. Otherwise, when $\mathcal{N}_m^T(u) \cap \mathcal{N}_m^B(v) = \emptyset$, $\Delta_T(u, v) = \Delta_B(u, v) = 1$. The reason to have two asymmetric metrics (in regard to the target and background graphs) instead of a symmetric one is that we want to choose those mappings which are the mutually best choices for the graphs. Again, a concrete example helps.

> [Dissimilarity.] Bob applied the dissimilarity metrics defined in Equations 5 and 6 to Figure 4 and got the results shown in Table 2.
> Bob first identified the tuples in Table 2 which has the smallest Δ_T and Δ_B in both its row and column. In this case, these tuples are (u_{*1}, v_{11}) and (u_{*3}, v_{12}). Since they are *from different rows and columns*, they do not conflict with each other. So Bob decided to map u_{*1} to v_{11} and u_{*3} to v_{12}.
> He then added $v_{*1} \leftrightarrow v_{11}$ and $v_{*3} \leftrightarrow v_{12}$ to the seed and moved on to the next iteration of identification.

3.2.2 Greedy Heuristic

Bob's story suggests a way of using the dissimilarity metrics defined in Equations 5 and 6 to iteratively grow the seed.

Since smaller dissimilarity implies better match, we identify those tuples in the table like Table 2 which has *smallest* Δ_T and Δ_B in both its row and column; these tuples are the mutually best matches between the target graph and the background graph. We then add the mappings corresponding to these tuples to the seed and move on to the next iteration.

We gloss over a subtlety in the above description: if there are *conflicts* in choice, i.e., there are multiple tuples satisfying the above criterion in a row or a column, which one shall we choose? Rather than randomly selecting a tuple, we select the tuple that *stands out* and add the corresponding match to the seed. If there is still a tie, these tuples are reckoned as indistinguishable under the dissimilarity metrics. To reduce incorrect identifications, we refrain from adding the mapping to the seed in these scenarios.

This boils down to the question of how to quantify the concept of "a tuple standing out among its peers." We define an *eccentricity* metric for this purpose in our algorithm. Let X be a group of numbers (the same number can occur multiple times). The *eccentricity* of a number $x \in X$ is defined as:

$$\mathcal{E}_X(x) = \begin{cases} \frac{\Delta_X(x)}{\sigma(X)\#_X(x)} & \text{if} \quad \sigma(X) \neq 0 \\ 0 & \text{if} \quad \sigma(X) = 0 \end{cases}. \tag{7}$$

in which $\Delta_X(x)$ is the absolute difference between x and its closest *different* value in X; $\#_X(x)$ is the *multitude* of x in X, i.e., the number of elements equal to x in X; $\sigma(X)$ is the standard deviation of X. The larger $\mathcal{E}_X(x)$ is, the more x stands out among X.

Therefore, if there are conflicts *in a row*, these tuples have the same Δ_T and Δ_B. For each such tuple, we collect the Δ_T and Δ_B *in the same column* into X_T and X_B respectively and compute $\mathcal{E}_{X_T}(\Delta_T)$ and $\mathcal{E}_{X_B}(\Delta_B)$. If there is a *unique* tuple with the *largest* $\mathcal{E}_{X_T}(\Delta_T)$ and $\mathcal{E}_{X_B}(\Delta_B)$, we pick it and add the corresponding mapping to the seed; otherwise, no mapping is added to the seed.

3.2.3 Revisiting

The dissimilarity metric and the greedy search algorithm for optimal combination are heuristic in nature. At an early stage with only a few seeds, there might be quite a few mapping candidates for a particular vertex in the background graph; we are very likely to pick a wrong mapping no matter which strategy is used in resolving the ambiguity. If left uncorrected, the incorrect mappings will propagate through the grow process and lead to large-scale mismatch.

We address this problem by providing a way to re-examine previous mapping decisions, given new evidences in the grow algorithm; we call this *revisiting*. More concretely, for each iteration, we consider all vertices which have at least one seed neighbor, i.e., those pairs of vertices on which the dissimilarity metrics in Equations 5 and 6 are well-defined.

We expect that the revisiting technique will increase the accuracy of the algorithm. The greedy heuristic with revisiting is summarized in Algorithm 3.

Algorithm 3 Grow.

```
Given the initial seeds V_S.
C = ∅
loop
    C_T ← {u ∈ V_T | u connects to V_S}
    C_B ← {v ∈ V_B | v connects to V_S}
    if (C_T, C_B) ∈ C then
        return V_S
    end if
    C ← C ∪ {(C_T, C_B)}
    for all (u, v) ∈ (C_T, C_B) do
        Compute Δ_T(u, v) and Δ_B(u, v).
    end for
    S ← {(u, v) | Δ_T(u, v) and Δ_B(u, v) are smallest among conflicts}
    for all (u, v) ∈ S do
        if (u, v) has no conflict in S or (u, v) has the uniquely largest eccentricity among conflicts in S then
            V_S ← V_S ∪ {(u, v)}
        end if
    end for
end loop
```

4 Experiments

We conducted a comparative study on the performance of the Seed-and-Grow algorithm by simulation on real-world social network datasets.

4.1 Setup

We used two datasets collected from different real-world social networks in our study.

The `Livejournal` dataset, which was collected from the friend relationship of the online journal service, LiveJournal, on December 9–11, 2006 [26], consists of 5.2 million vertices and 72 million links. The links are directed. As previously discussed at the end of Section 2, we conducted the experiments with the more difficult setting of an undirected graph. We retained an undirected link between two vertices if there was a directed link in either direction.

The other dataset, `emailWeek`[2], consists of 200 vertices and 1,676 links. This dataset, by its nature, is undirected.

Using datasets collected from different underlying social networks helped to reduce bias induced by the idiosyncrasy of a particular network in performance measurements.

The performance of the grow algorithm was measured by its ability to identify the anonymous vertices in the target graph. We derived the target and background graphs from each dataset and used their shared vertices as the *ground truth* to measure against.

More precisely, we derived the graphs with the following procedure. First, we chose a connected subgraph with $N_\cap$ vertices from the dataset, which served as a *shared portion* of the background and target graphs. We then picked other two sets of vertices (different from the previous $N_\cap$ vertices) with $N_B - N_\cap$ and $N_T - N_\cap$ vertices, respectively, and combined with shared portion graph to obtain the background graph (with N_B vertices) and the target graph (with N_T vertices). After this, N_S ($N_S < N_\cap$ and not necessarily connected) vertices were chosen from the shared portion to serve as the initial seed. Finally, random edges were added to the target graph to simulate the difference between the target and background graphs.

The motivation for adopting such a procedure was to simulate a more realistic scenario. The attacker had a (connected) background graph with N_B vertices and an anonymous target graph with N_T vertices. Apart from the N_S initial seed, the overlap of these two graphs (with

2. http://www.infovis-wiki.net/index.php/Social_Network_Generation.

TABLE 3
The estimate of essentially different constructions for a fingerprint graph G_F with n vertices produced by Algorithm 1.

n	10	11	12	13
estimate	1.89×10^6	9.70×10^7	9.03×10^8	1.54×10^{11}

$N_\cap$ vertices) might well be partial. The desirable behavior of an identification algorithm was to stop as soon as the vertices in the shared portion had been identified. Since the background graph was an unperturbed graph the attacker obtained from elsewhere, we opted to perturb the target graph to simulate the difference between these two graphs. We perturbed by addition rather than deletion of edges to avoid fragmenting the target graph into disconnected pieces, which would create a false impression of early stopping in simulation.

4.2 Seed

The Seed construction (Algorithm 1) and recovery (Algorithm 2) algorithms ensure that, once the fingerprint graph G_F is successfully recovered, the initial seed V_S can be unambiguously identified. Therefore, the seed construction depends on G_F being uniquely recovered from the released target graph.

We randomly generated a number of modest-sized fingerprint graphs with 10 to 20 vertices and planted them into the `Livejournal` dataset with Algorithm 1. We were able to uniquely recover them from the resulted graph with Algorithm 2 without exception.

To explain this result, we made the following estimation on the number of essentially different (i.e., with different ordered internal degree sequence $\mathcal{S}_D$) constructions produced by Algorithm 1.

For a fingerprint graph G_F with n vertices, there are $n-1$ vertices beside the head node v_h. There are $(n-1)(n-2)/2$ pairs among the $n-1$ vertices; the edge between each pair of vertices can be either present or absent. Therefore, there are $2^{(n-1)(n-2)/2}$ different fingerprint graphs.

However, some of them are considered the same by Algorithm 1. For example, the ordered internal degree sequence $\mathcal{S}_D = \langle 2,2,2,3,3,4\rangle$ in Figure 3. There are 3, 2, and 1 vertices with an internal degree of 2, 3, and 4, respectively; hence, there are $\binom{6}{3}\binom{3}{2}\binom{1}{1}$ different fingerprint graphs with the same ordered internal degrees sequence.

For any ordered internal degree sequence $\mathcal{S}_D$, there are *at most* $\binom{n-1}{1}\binom{n-2}{1}\cdots\binom{2}{1}\binom{1}{1} = (n-1)!$ fingerprint graphs with n vertices. The ordered internal degree sequence divides all fingerprint graphs into equivalent classes. Therefore, the number of essentially different constructions produced by Algorithm 1 is:

$$\frac{2^{(n-1)(n-2)/2}}{(n-1)!}.$$

Table 3 shows this estimate for a few different fingerprint graph sizes. From this, we can understand the reason for the high probability for successful fingerprint graph recovery, even in a large graph like `Livejournal` with 5.2×10^6 vertices: there are so many ways to construct essentially different fingerprint graphs.

4.3 Grow

We compared our grow algorithm with the one proposed by Narayanan and Shmatikov [2]. There is a mandatory threshold parameter, which controls the probing aggressiveness, in their algorithm. Lacking a quantitative guideline to choose this parameter in [2], we experimented with different values and found that, with an increasing threshold, more nodes were identified but the accuracy decreased accordingly. Therefore, we used two different thresholds, which established a performance envelope for the Narayanan algorithm. The result was two variants of the algorithm: an aggressive one (with a threshold of 0.0001) and a conservative one (with a threshold of 1). The difference lay in the tolerance to the ambiguities in matching: the aggressive one might declare a mapping in a case where the conservative one would deem too ambiguous.

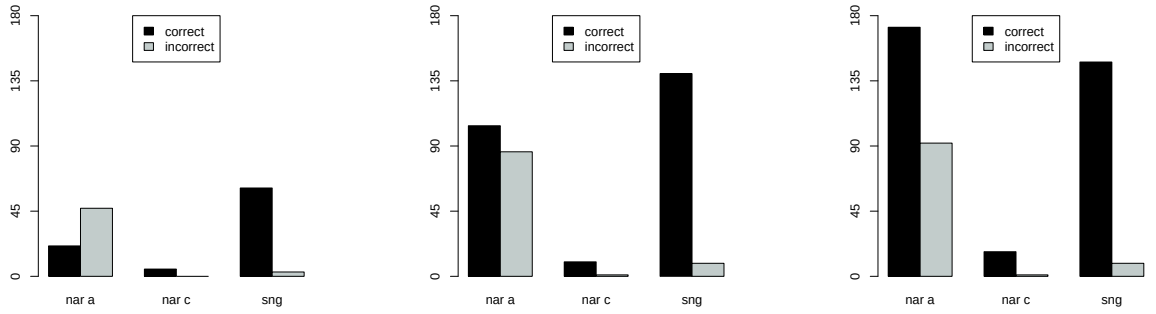
We perceive such an arbitrary parameter, lacking a quantitative guideline, as a major drawback of the Narayanan algorithm: a user of the algorithm *must* decide on the parameter without knowing how much accuracy is sacrificed for better effectiveness (the number of identified nodes). In contrast, our grow algorithm has no such parameter and, as demonstrated by the experiments, finds a good balance between effectiveness and accuracy.

To account for the bias on the performance measurement of a particular graph setting, for each target/background graph pair, we ran multiple simulations with different initial seeds and took the average as the performance measurement. We focused our simulations on graphs with hundreds of vertices, which are big enough to make the identification non-trivial. More precisely, we chose $(N_C = 400 + N_S, N_T = 600 + N_S, N_B = 600 + N_S)$ for `Livejournal` and $(N_C = 100 + N_S, N_T = 125 + N_S, N_B = 125 + N_S)$ for `emailWeek`. In other words, the ideal result is to correctly identify $400 + N_S$ nodes for `Livejournal` and $100 + N_S$ nodes for `emailWeek` where N_S is the size of initial seed.

4.3.1 Initial Seed Size

Recent literature on interaction-based social graphs (e.g., the social graph in the motivating scenario) singles out the attacker's interaction budget as the major limitation to attack effectiveness [33]. The limitation translates to 1) the initial seed size and 2) the number of links between the fingerprint graph and the initial seed. Our seed algorithm resolves the latter issue by guaranteeing unambiguous identification of the initial seed, regardless of link numbers. As shown below, our grow algorithm resolves the former issue by working well with a small initial seed.

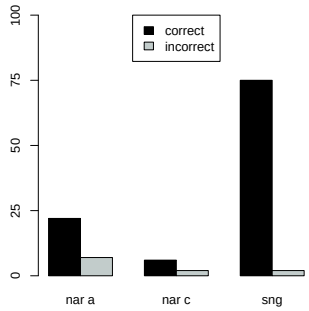
Figure 5 shows the grow performance with different initial seed sizes. To simulate the more realistic case that



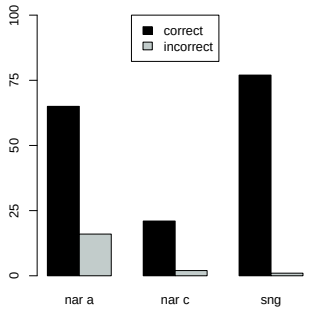
(a) `Livejournal`, seed: 5, pertb: 0.5%

(b) `Livejournal`, seed: 10, pertb: 0.5%

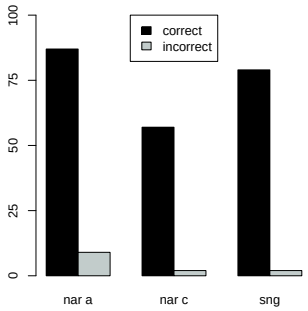
(c) `Livejournal`, seed: 15, pertb: 0.5%



(d) `emailWeek`, seed: 5, pertb: 0.5%

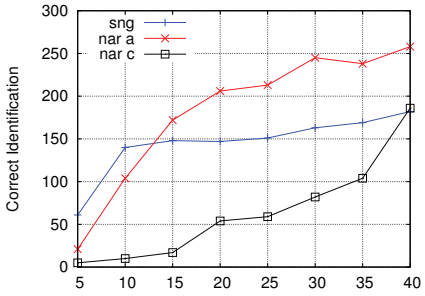


(e) `emailWeek`, seed: 10, pertb: 0.5%

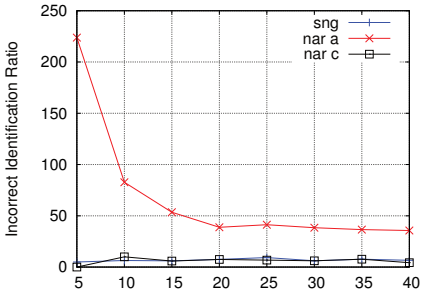


(f) `emailWeek`, seed: 15, pertb: 0.5%

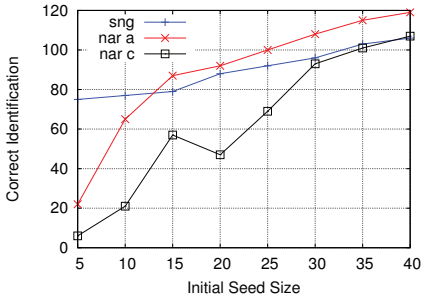
Fig. 5. Grow performance with different initial seed sizes. An edge perturbation of 0.5% is introduced to simulate a more realistic scenario. (a), (b), and (c) are from `Livejournal`; (d), (e), and (f) are from `emailWeek`.



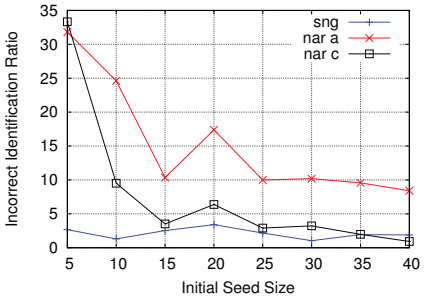
(a) `Livejournal`, correct identifications.



(b) `Livejournal`, incorrect identifications.



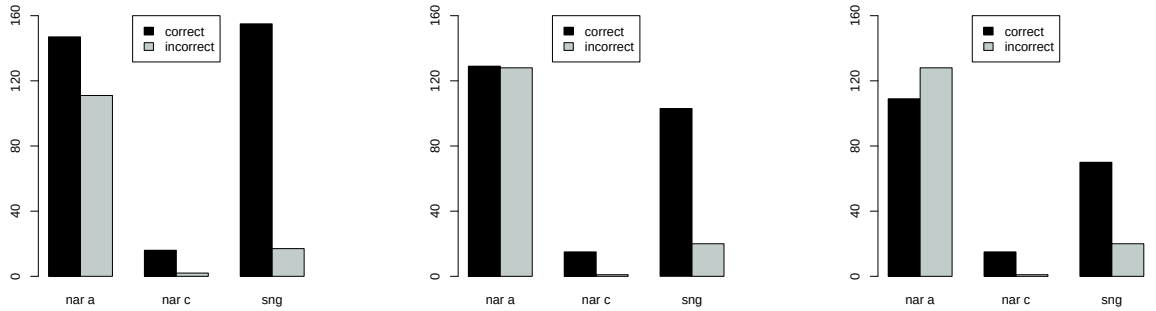
(c) `emailWeek`, correct identifications.



(d) `emailWeek`, incorrect identifications.

Fig. 6. Grow performance with different initial seed sizes on a larger scale than Figure 5. An edge perturbation of 0.5% is introduced to simulate a more realistic scenario. (a) and (b) are from `Livejournal`; (c) and (d) are from `emailWeek`.

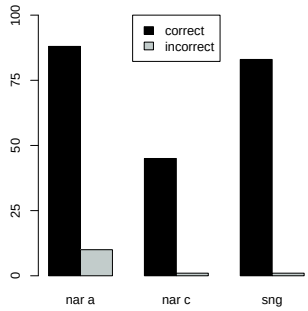
the target and background graphs are from different sources, and therefore might differ even among the same group of vertices, we introduced an *edge perturbation* of 0.5%, i.e., we added 0.5% of the all of the edges in the



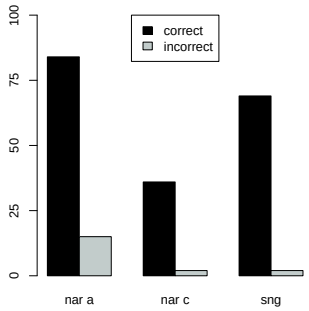
(a) `Livejournal`, seed: 15, pertb: 0.5%

(b) `Livejournal`, seed: 15, pertb: 1%

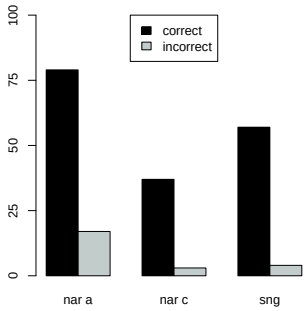
(c) `Livejournal`, seed: 15, pertb: 1.5%



(d) `emailWeek`, seed: 15, pertb: 0.5%

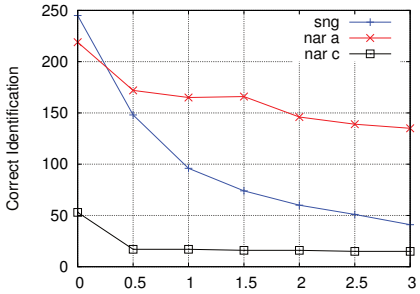


(e) `emailWeek`, seed: 15, pertb: 1%

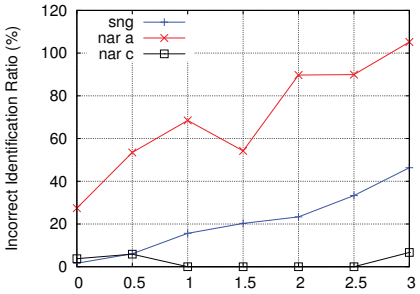


(f) `emailWeek`, seed: 15, pertb: 1.5%

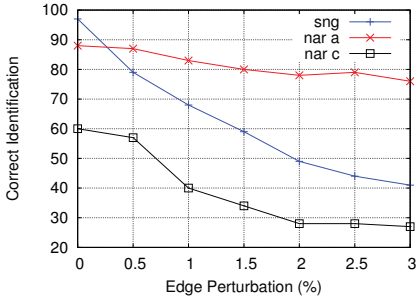
Fig. 7. Grow performance with different edge perturbation percentage. The initial seed size is 15 for both datasets. (a), (b), and (c) are from `Livejournal`; (d), (e), and (f) are from `emailWeek`.



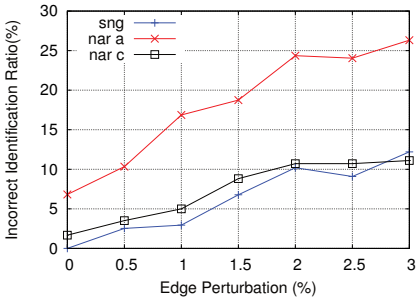
(a) `Livejournal`, correct identifications.



(b) `Livejournal`, incorrect identifications.



(c) `emailWeek`, correct identifications.



(d) `emailWeek`, incorrect identifications.

Fig. 8. Grow performance with different edge perturbation percentage on a larger scale than Figure 7. The initial seed size is 15 for both datasets. (a) and (b) are from `Livejournal`; (c) and (d) are from `emailWeek`.

target graph.

We note a few points for Figure 5.

- More nodes are correctly identified with increasing initial seed size for both Seed-and-Grow and Narayanan.
- Seed-and-Grow is better than (or at least compa-

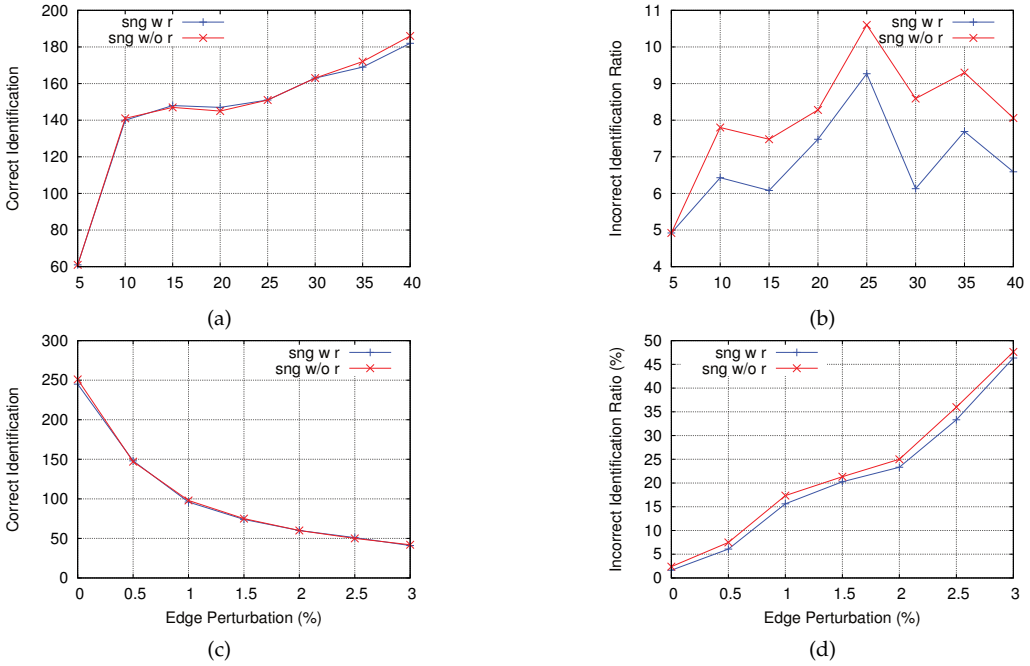


Fig. 9. Grow performance comparison of Seed-and-Grow with (sng w r) and without (sng w/o r) revisiting heuristic on the `Livejournal` dataset. The edge perturbation for (a) and (b) is 0.5%. The initial seed size is 15 for (c) and (d).

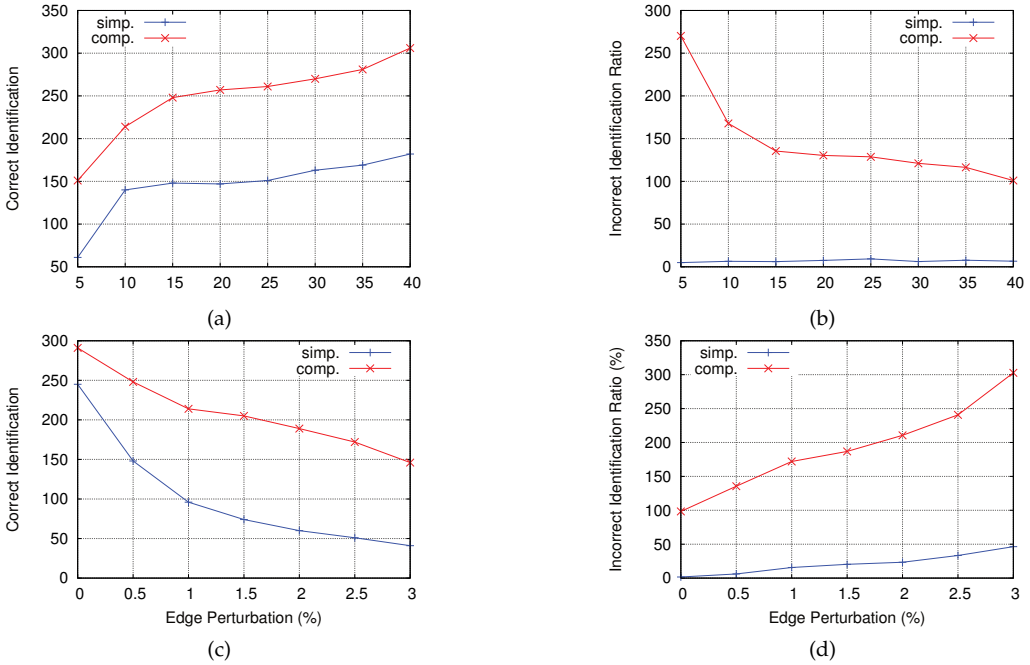


Fig. 10. Grow performance comparison of Seed-and-Grow with different dissimilarity designs (simp.: simple yet accurate; comp.: complex yet inaccurate) for the `Livejournal` dataset. The edge perturbation for (a) and (b) is 0.5%. The initial seed size is 15 for (c) and (d).

rable to) the aggressive Narayanan in terms of number of correct identifications, and is superior when comparing with conservative Narayanan. For example, in `Livejournal`, conservative Narayanan stops almost immediately (the correct identification statistics shown in Figure 5 includes the initial seed). In contrast, even for very small initial seed of 5 nodes, Seed-and-Grow correctly identifies averagely 61 nodes for `Livejournal` and 75 nodes for `emailWeek`, while only incorrectly identifying 1 nodes on average.

- Though aggressive Narayanan correctly identifies more nodes as seed size grows, the number of incorrect identification grows accordingly. This is particularly evident in `Livejournal`. In contrast, the incorrect identification number for Seed-and-Grow

remains constant in `emailWeek` and grows very slowly in `Livejournal`; in either case, the number of correct identifications is considerably higher for Seed-and-Grow than for aggressive Narayanan.

An ideal grow algorithm should be both effective and accurate. Effectiveness is measured by the number of correct identifications; accuracy is measured by the ratio of incorrect identifications over that of correct identifications (the lower the ratio, the higher the accuracy). Figure 5 shows Seed-and-Grow was: 1) comparable to aggressive Narayanan in terms of effectiveness, while better in terms of accuracy; 2) comparable to conservative Narayanan in terms of accuracy, while better in terms of effectiveness.

Figure 6 shows, on a larger scale, the comparison in Figures 5 and 7 in a slightly different form. The previous observations on algorithm performance, though less stark, still hold for the larger seeds. Since the seed size translates to attacker's cost, Seed-and-Grow, which is both effective and accurate for a small seed, is desirable.

It is arguable that, with a "proper" threshold, Narayanan will show the same, or even superior performance, than Seed-and-Grow. However, lacking any quantitative guideline, such a proper threshold is hard, if not impossible, to find for the vast array of graphs the identification algorithm applies to. Even if one can find such a threshold, it is unclear that its performance will be superior to that of Seed-and-Grow. In contrast, Seed-and-Grow has no such arbitrary parameter. The point is that Seed-and-Grow finds a sensible balance between effectiveness and accuracy without prior knowledge as in Narayanan.

4.3.2 Edge Perturbation

The impact of edge perturbations on the grow performance is shown in Figure 7. The initial seed size was 15.

Correct identifications decreased with a larger edge perturbation percentage for all algorithms. Incorrect identifications increased with edge perturbation for aggressive Narayanan, while remaining at a constant level for Seed-and-Grow and conservative Narayanan.

Seed-and-Grow was more effective than conservative Narayanan in all settings. Although aggressive Narayanan was more effective than Seed-and-Grow for larger perturbation percentage, it came with a much higher cost in accuracy; for `Livejournal`, aggressive Narayanan made more incorrect identifications than correct ones. In contrast, the number of incorrect identifications for Seed-and-Grow remain almost constant with different perturbation percentages. Figure 8 shows the results on a larger scale: Seed-and-Grow was able to grow the seed larger (unlike conservative Narayanan) while maintaining a relatively high accuracy (unlike aggressive Narayanan).

4.3.3 Revisiting

To verify our expectation for the revisiting heuristic to improve performance, we compared the Seed-and-Grow algorithm with and without the revisiting heuristic. The results on the `Livejournal` are shown in Figure 9; the results on `emailWeek` are similar but omitted due to the space constraint.

The results indicate that the revisiting heuristic improves (i.e. reduces) the incorrect identification ratio by 2% to 3% without sacrificing the number of correct identifications. This confirms our conjecture that it is worthwhile to revisit previous mappings.

A high accuracy (i.e., a high percentage of correct identifications) is desirable, even at a reasonable cost of effectiveness (fewer nodes identified). This is because, lacking the knowledge about whether or not an identification is correct, accuracy corresponds to the user's *confidence* in the identification result. For example, in Figure 7c, even though aggressive Narayanan correctly identified 166 nodes on average, while Seed-and-Grow only correctly identified 74 nodes on average, the former incorrectly identified 90 nodes on average, while the latter only incorrectly identified 15 nodes on average. Without knowing which nodes were correctly identified, a user had less than 65% confidence in the results of aggressive Narayanan, while having more than 80% confidence in the results of Seed-and-Grow.

On reflection, we attribute the relatively high accuracy of Seed-and-Grow to the conservative design in our grow algorithm (Algorithm 3). More specifically, we add a mapping to the seed (i.e., grow the seed) if and only if 1.) it is the mutually best choice for the pair of nodes under the dissimilarity metric and 2.) it stands out among alternative choices in the sense that it has no tie under the eccentricity metric. Besides, the algorithm further improves accuracy by revisiting earlier mappings in light of new mappings.

4.3.4 Dissimilarity

In Section 3.2.1, we mentioned that the simple dissimilarity metrics defined by Equations (5) and (6) were the result of evolving from an earlier, more complex, and apparently more comprehensive design specified by Equations (1)–(4). Our decision to adopt the simple design was based on the simulation results shown in Figure 10: Although the simple design is less effective than the complex one in terms of number of correct identifications, the former is much more accurate than the latter. For the same reason as in Section 4.3.3, the higher accuracy of the simple design is desirable. Therefore, the simple design was adopted in the Seed-and-Grow algorithm, as discussed in Section 3.2.1.

5 Summary

We propose an algorithm, *Seed-and-Grow*, to identify users from an anonymized social graph. Our algorithm exploits the increasing overlapping user-bases among services and is based solely on social graph structure. The algorithm first identifies a seed sub-graph, either planted by an attacker or divulged by collusion of a small group of users, and then grows the seed larger

based on the attacker's existing knowledge of the users' social relations. We identify and relax implicit assumptions for unambiguous seed identification taken by previous works, eliminate arbitrary parameters in grow algorithm, and demonstrate the superior performance over previous works in terms of identification effectiveness and accuracy by simulations on real-world-collected social-network datasets.

REFERENCES

[1] B. Krishnamurthy and C. E. Wills, "Characterizing privacy in online social networks," in *Proc. ACM WOSN*, 2008.
[2] A. Narayanan and V. Shmatikov, "De-anonymizing social networks," in *Proc. IEEE S&P*, 2009.
[3] L. Backstrom, C. Dwork, and J. Kleinberg, "Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography," in *Proc. ACM WWW*, 2007.
[4] M. Hay, G. Miklau, D. Jensen, P. Weis, and S. Srivastava, "Anonymizing social networks," Univ. Massachusetts, Amherst, Tech. Rep., 2007.
[5] E. Zheleva and L. Getoor, "Preserving the privacy of sensitive relationships in graph data," in *Proc. ACM SIGKDD*, 2007.
[6] A. Korolova, R. Motwani, S. Nabar, and Y. Xu, "Link privacy in social networks," in *Proc. ACM CIKM*, 2008.
[7] B. Zhou and J. Pei, "Preserving privacy in social networks against neighborhood attacks," in *Proc. Intl. Conf. on Data Engineering (ICDE)*. IEEE, 2008.
[8] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis, "Resisting structural re-identification in anonymized social networks," *VLDB Endowment*, vol. 1, no. 1, pp. 102–114, 2008.
[9] J. Scott, *Social network analysis: a handbook*. SAGE Publications, 2000.
[10] K. LeFevre, D. DeWitt, and R. Ramakrishnan, "Incognito: efficient full-domain k-anonymity," in *Proc. ACM ICMD*, 2005.
[11] B. Zhou, J. Pei, and W. Luk, "A brief survey on anonymization techniques for privacy preserving publishing of social network data," *ACM SIGKDD Explorations Newsletter*, vol. 10, no. 2, pp. 12–22, 2008.
[12] A. Mislove, H. Koppula, K. Gummadi, P. Druschel, and B. Bhattacharjee, "Growth of the flickr social network," in *Proc. WOSN*. ACM, 2008.
[13] R. Xiang, J. Neville, and M. Rogati, "Modeling relationship strength in online social networks," in *Proc. ACM WWW*, 2010.
[14] J. Douceur, "The sybil attack," *LNCS*, vol. 2429, pp. 251–260, 2002.
[15] N. Tran, B. Min, J. Li, and L. Subramanian, "Sybil-resilient online content voting," in *Proc. USENIX NSDI*, 2009.
[16] S. Park, B. Aslam, D. Turgut, and C. Zou, "Defense against sybil attack in vehicular ad hoc network based on roadside unit support," in *Proc. IEEE MILCOM*, 2009.
[17] C. Lesniewski-Laas and M. Kaashoek, "Whanau: A sybil-proof distributed hash table," in *Proc. USENIX NSDI*, 2010.
[18] C. Chen, X. Wang, W. Han, and B. Zang, "A robust detection of the sybil attack in urban vanets," in *Proc. IEEE ICDCS*, 2009.
[19] H. Yu, M. Kaminsky, P. Gibbons, and A. Flaxman, "Sybilguard: defending against sybil attacks via social networks," *ACM SIGCOMM CCR*, vol. 36, no. 4, pp. 267–278, 2006.
[20] H. Yu, P. Gibbons, M. Kaminsky, and F. Xiao, "Sybillimit: A near-optimal social network defense against sybil attacks," in *Proc. IEEE S&P*, 2008.
[21] B. Viswanath, A. Post, K. Gummadi, and A. Mislove, "An analysis of social network-based sybil defenses," *ACM SIGCOMM CCR*, vol. 40, no. 4, pp. 363–374, 2010.
[22] W. Wei, F. Xu, C. Tan, and Q. Li, "SybilDefender: Defend against sybil attacks in large social networks," in *Proc. IEEE INFOCOM*, 2012.
[23] S. Sorlin and C. Solnon, "Reactive tabu search for measuring graph similarity," *LNCS*, vol. 3434, pp. 172–182, 2005.
[24] P. Erdös and A. Rényi, "On random graphs," *Publicationes Mathematicae*, vol. 6, no. 26, pp. 290–297, 1959.
[25] R. Kumar, J. Novak, and A. Tomkins, "Structure and evolution of online social networks," in *Proc. ACM SIGKDD*, 2006.
[26] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *Proc. ACM IMC*, 2007.
[27] J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney, "Statistical properties of community structure in large social and information networks," in *Proc. ACM WWW*, 2008.
[28] M. Porter, J. Onnela, and P. Mucha, "Communities in networks," *Notices of the AMS*, vol. 56, no. 9, pp. 1082–1097, 2009.
[29] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins, "Microscopic evolution of social networks," in *Proc. ACM SIGKDD*, 2008.
[30] A. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.
[31] D. Soares, C. Tsallis, A. Mariz, and L. Silva, "Preferential attachment growth model and nonextensive statistical mechanics," *Europhysics Letters*, vol. 70, p. 70, 2005.
[32] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *Proc. IEEE S&P*, 2008.
[33] C. Wilson, B. Boe, A. Sala, K. Puttaswamy, and B. Zhao, "User interactions in social networks and their implications," in *Proc. ACM EuroSys*, 2009.

Wei Peng is a PhD student with the Department of Computer and Information Science of Indiana University-Purdue University Indianapolis. His co-advisors are Dr. Feng Li and Dr. Xukai Zou. He has worked on problems on delay-tolerant networks, security, privacy, and social networks. His research vision is to explore human factors in computing and networking and, in turn, make them more human friendly.

Feng Li received his Ph.D. in Computer Science from Florida Atlantic University in Aug. 2009. His Ph.D. advisor is Dr. Jie Wu. He joined the Department of Computer, Information, and Leadership Technology at Indiana University-Purdue University Indianapolis (IUPUI) as an assistant professor in Aug. 2009. His research interests include the areas of wireless networks and mobile computing, security, and trust management. He has published more than 30 papers in conferences and journals.

Xukai Zou is a faculty member with the Department of Computer and Information Sciences at Indiana University-Purdue University Indianapolis. He completed his PhD Degree in Computer Science from University of Nebraska-Lincoln. His current research focus is Applied Cryptography, Network Security, and Communication Networks. His research has been supported by NSF, the Department of Veterans Affairs and Industry such as Cisco.

Jie Wu is chair and professor in the Department of Computer and Information Sciences at Temple University. Prior to joining Temple University, he was a program director at the National Science Foundation. His research interests include wireless networks, mobile computing, routing protocols, fault-tolerant computing, and interconnection networks. Dr. Wu publications include over 550 papers in scholarly journals and conference proceedings. Additionally, he has served on several editorial boards, including IEEE Transactions on Computers and Journal of Parallel and Distributed Computing. Dr. Wu was general co-chair for IEEE MASS 2006, IEEE IPDPS2008 and DCOSS 2009 and is the program co-chair for IEEE INFOCOMM 2011. He served as an IEEE Computer Society distinguished visitor. Currently, he is the chair for the IEEE Technical Committee on Distributed Processing (TCDP), an ACM distinguished speaker and a Fellow of the IEEE. Dr. Wu is the recipient of 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.