Towards Graph Watermarks

Xiaohan Zhao, Qingyun Liu, Haitao Zheng and Ben Y. Zhao
Computer Science, UC Santa Barbara
{xiaohanzhao, qingyun_liu, htzheng, ravenben}@cs.ucsb.edu

ABSTRACT

From network topologies to online social networks, many of today's most sensitive datasets are captured in large graphs. A significant challenge facing the data owners is how to share sensitive graphs with collaborators or authorized users, *e.g.* ISP's network topology graphs with a third party networking equipment vendor. Current tools can provide limited node or edge privacy, but significantly modify the graph reducing its utility.

In this work, we propose a new alternative in the form of *graph watermarks*. Graph watermarks are small graphs tailor-made for a given graph dataset, a secure graph key, and a secure user key. To share a sensitive graph G with a collaborator C, the owner generates a watermark graph W using G, the graph key, and C's key as input, and embeds W into G to form G'. If G' is leaked by C, its owner can reliably determine if the watermark W generated for C does in fact reside inside G', thereby proving C is responsible for the leak. Graph watermarks serve both as a deterrent against data leakage and a method of recourse after a leak. We provide robust schemes for embedding and extracting watermarks, and use analysis and experiments on large real graphs to show that they are unique and difficult to forge. We study the robustness of graph watermarks against both single and powerful colluding attacker models, then propose and evaluate mechanisms to dramatically improve resilience.

1. INTRODUCTION

Many of today's most sensitive datasets are captured in large graphs. Such datasets can include maps of autonomous systems in the Internet, social networks representing billions of friendships, or connected records of patent citations. Controlling access to these datasets is a difficult challenge. More specifically, it is often the case that owners of large graph datasets would like to share access to them to a fixed set of entities without the data leaking into the public domain. For example, an ISP may be required to share detailed network topology graphs with a third party networking equipment vendor, with a strict agreement that access to these sensitive graphs must be limited to authorized personnel only. Similarly, a large social network like Facebook or LinkedIn may choose to share portions of its social graph data with trusted academic collaborators, but clearly want to prevent their leakage into the broader research community.

One option is to focus on building strong access control mechanisms to prevent data leakage beyond authorized parties. Yet in most scenarios, including both examples above, data owners cannot restrict physical access to the data, and have limited control once the data is shared with the trusted collaborator. It is also the case that no matter how well access control systems are designed, they are never foolproof, and often fall prey to attacks on the human element, *i.e.* social engineering. Another option is to modify portions of the data to reduce the impact of potential data leakages. This has the downside of making the data inherently noisy and inaccurate, and still can be overcome by data reconstruction or de-anonymization attacks using external input [27]. Finally, these schemes are hard to justify, in part because it is very difficult to quantify the level of protection they provide.

In this work, we propose a new alternative in the form of *graph watermarks*. Intuitively, watermarks are small, often imperceptible changes to data that are difficult to remove, and serve to associate some metadata to a particular dataset. They are used successfully today to limit data piracy by music vendors such as Apple and Walmart, who embed a user's personal information into a music file at the time of purchase/download [3]. Should the purchased music be leaked onto music sharing networks, it is easy for Apple to track down the user who was responsible for the leak. In our context, graph watermarks work in a similar way, by securely identifying a copy of a graph with its "authorized user." Should a shared graph dataset be leaked and discovered later in public domains (on BitTorrent perhaps), the data owner can extract watermark from the leaked copy and use it as proof to seek damages against the collaborator responsible for the leak. While not a panacea, graph watermarks can provide additional level of protection for data owners who want to or must share their data, and perhaps encourage risk-averse data owners to share potentially sensitive graph data, *e.g.* encourage LinkedIn to share social graphs with academic collaborators.

To be effective, a graph watermark system needs to provide several key properties. *First*, graph watermarks should be relatively small compared to the graph dataset itself. This has two direct consequences: the watermark will be difficult to detect (and remove) by potential attackers, and adding the watermark to the graph has minimal impact on the graph structure and its utility. *Second*, watermarks should be difficult to forge and should not occur naturally in graphs, ensuring that the presence of a valid watermark can be securely associated with some user, *i.e.* non-repudiation. *Third,* both the embedding and extraction of watermarks should be efficient, even for extremely large graph datasets with billions of nodes

COSN'15, November 2–3, 2015, Palo Alto, California, USA.

ACM 978-1-4503-3951-3/15/11 ...$15.00.
DOI: http://dx.doi.org/10.1145/2817946.2817956.

and edges. *Finally,* our goal is to design a watermark system that works in any application context involving graphs. Therefore, we make no assumptions about the presence of metadata. Instead, our system must function for "barebones" graphs, *i.e.* symmetric, unweighted graphs with no node labels or edge weights.

In this paper, we present initial results of our efforts towards the design of a scalable and robust graph watermark system. Highlights of our work can be organized into the following key contributions.

- First, we identify the goals and requirements of a graph watermark system. We also describe an initial design of a graph watermark system that efficiently embeds watermarks into and extracts them out of large graphs. Graph watermarks are uniquely generated based on a user private key, a secure graph key, and the graph they are applied to. We describe constraints on its applicability, and identify examples of graphs where watermarks cannot achieve desirable levels of key properties such as uniqueness.
- Second, we provide a strict proof of uniqueness of graph watermarks, showing that it is extremely difficult for attackers to forge watermarks.
- Third, we evaluate our watermarks in term of distortion, uniqueness, and efficiency on several large graph datasets.
- Fourth, we identify two attack models, describe additional features to boost robustness, and evaluate them under realistic conditions.

To the best of our knowledge, our work is the first practical proposal for applying watermarks to graph data. We believe graph watermarks are a useful tool suitable for a wide range of applications from tracking data leaks to data authentication. Our work identifies the problem and defines an initial groundwork, setting the stage for follow-up work to improve robustness against a range of stronger attacks.

2. BACKGROUND AND RELATED WORK

In this section, we provide background and related work on graph privacy and watermark techniques in applications.

Graph Privacy. Graph privacy is a significant problem that has been magnified by the arrival of large graphs containing sensitive data, *e.g.* online social graphs or mobile call graphs. Recent studies [4, 27] show that deanonymization attacks can defeat most common anonymization techniques.

A variety of solutions have been proposed, ranging from anonymization tools that defend against specific structural attacks, or more attack-agnostic defenses. To protect node- or edge-privacy against specific, known attacks, techniques utilize variants of *k-anonymization* to produce structural redundancy at the granularity of subgraphs, neighborhoods or single nodes [23, 46, 12, 48]. Alternatively, randomization provides privacy protection by randomly adding, deleting, or switching edges [10, 44]. Others partition the nodes and then describe the graph at the level of partitions to avoid structural re-identification [11]. Finally, a different approach is taken by producing model-driven synthetic graphs that replicate key structural properties of the original graphs [35]. One extension of this work utilizes differential privacy techniques to provide a tunable accuracy vs. privacy tradeoff [36].

Our goals are quite different from prior work on graph anonymization, meant to protect data before its public release. We are concerned with scenarios where graph data is shared between its owner and groups of trusted collaborators, *e.g.* third party network vendors analyzing an ISP's network topology, or Facebook sharing a graph with a group of academic researchers. The ideal goal in these scenarios is to ensure the shared data does not leak into the wild. Once data is shared with collaborators, reliable tools that can track leaked data back to its source serve as an excellent deterrent. Watermarking techniques have addressed similar problems in other contexts, and we briefly describe them here.

Background on Digital Watermarks. Watermarking is the process of embedding specialized metadata into multimedia content [14]. The embedded *watermark* is later extracted from the file and used to identify the source or owner of the content. These systems include an embedding component and an extraction component. The embedding component takes three inputs: a watermark, the original data, and a key, aiming to embed the watermark with minimum impact on the data. The key is used as a parameter to generate a unique watermark for a specific user, and is kept confidential by the data owner. Extraction takes as input the watermarked data, the key, and possibly a copy of the original data. Extraction can directly produce the embedded watermark or a confidence measure of whether it is present.

Watermarking is widely used today to protect intellectual property. Significant work has been done in digital watermarking, particularly image watermarking [37, 24, 5, 34, 42]. Watermark techniques [29, 30] have been studied to protect the abuse of digital vector maps. Watermarks have also been used to protect software copyrights [47, 7], by adding spurious execution paths in the code that would not be triggered by normal inputs [39]. Moreover, watermark algorithms have been proposed for relational datasets [1, 22, 13]. Much of this has focused on modifying numeric attributes of relations, using the primary key attribute as an indicator of watermark locations, assuming that the primary key attribute does not change. Finally, watermarks, in the form of minute changes, have been applied to protect circuit designs in the semiconductor industry [31, 41].

3. GOALS AND ATTACK MODELS

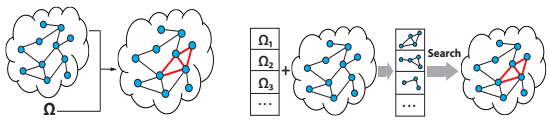
To set the context for the design of our graph watermark system, we need to first clearly define the attack models we target, and use them to guide our design goals.

Graph watermarks at a glance. At a high level, we envision the graph watermark process to be simple and lightweight, as pictured in Figure 1. Embedding a watermark involves *overlaying* the original graph dataset (G) with a small subgraph (W) generated using the original graph and a secret random generator seed (Ω). Embedding the watermark simply means adding or deleting edges between existing nodes in the original graph G, based on the watermark subgraph W. Each authorized user i receives only a watermarked graph customized for her, generated using a random seed Ω_i securely associated with her. The seed is generated through cooperation of her private key and a key securely associated with the original graph.

If and when the owner detects a leaked version of the dataset, the owner takes the leaked graph, and "extracts the watermark," by iteratively producing all known watermark subgraphs W_i associated with G and each of the seeds Ω_i associated with an authorized user. The "extraction" process is actually a matching process where the data owner can conclusively identify the source of the leaked data, by locating the matching W_i in the leaked graph.

In our model of potential attackers and threats, we assume that attackers have access to the watermarked graph, but not the original G. Clearly, if an attacker is able to obtain the unaltered G, then watermarks are no longer necessary.

Attack Models. The attackers' goal is to destroy or remove graph watermarks while preserving the original graph. Watermarks are designed to protect the overall integrity of the graph data. Thus



(a) Embedding (b) Extraction

Figure 1: Embedding and extracting graph watermarks.

we do not consider scenarios where the attackers sample the graph or distort it significantly to remove the watermark. Under these constraints, we consider two practical attack models below.

- *Single Attacker Model.* For a single attacker with access to one watermarked graph, it will be extremely difficult to detect the embedded watermark. Without the key associated with another user, forging a watermark is also impractical. Instead, their best attack is to disrupt any potential watermarks by adding or deleting nodes or edges.
- *Collusion Attack Model.* If multiple attackers join their efforts, they can recover the original graph by comparing multiple watermarked graphs, identifying the differences (*i.e.* watermarks), and removing them.

Design Goals. The attack models help us define the key characteristics required for a graph watermarking system.

- *Low distortion.* The addition of watermarks should have a small impact on overall structure of the original graph. This preserves the utility of the graph datasets.
- *Robust to modifications.* Watermarks should be robust to modification attacks on watermarked graphs, *i.e.* watermarks should remain detectable and extractable with high probability, even after the graph has been modified.
- *Low false positives.* It is extremely unlikely for our system to successfully identify a valid watermark W_i in an unwatermarked graph or a graph watermarked by W_j where $i \neq j$. When we embed a single watermark (Section 4), we also refer to this property as *watermark uniqueness*.

Within the constraints defined above, designing a graph watermark system is quite challenging, for several reasons. First, the subgraph that represents the watermark must be relatively "unique," *i.e.* it is highly unlikely to occur naturally, or intentionally through forgery. A second, contrasting goal is that the watermark should not change the underlying graph significantly (low distortion), or be easily detected. Walking the fine line between this and properties of "uniqueness" likely means we have to restrict the set of graphs which can be watermarked, *i.e.* for some graphs, it will be impossible to find a hard to detect watermark that does not occur easily in graphs. Finally, since any leaked graph can have all metadata stripped or modified, watermark embedding and extraction algorithms must function without any labels or identifiers. Note that the problem of subgraph matching is known to be NP-complete [8].

4. BASIC WATERMARK DESIGN

We now describe the basic design of our graph watermarking system. The basic design seeks to embed and extract watermarks on graphs to achieve watermark uniqueness while minimizing distortion on graph structure. Our design has two key components:

- **Watermark embedding**: The data owner holds a graph key K^G associated with a graph G known only to her. Each user i generates its public-private cryptographic key pair $< K^i_{pub}, K^i_{priv} >$ through a standard public-key algorithm [25], where K^i_{pub} is user i's public key and K^i_{priv} is its corresponding private key. To share the graph G with user i, the system combines input from user i's digital signature $K^i_{priv}(T)$ and graph key K^G to form a random generator seed Ω_i, and use Ω_i to generate a watermark graph W_i for graph G. The system embeds W_i into G by selecting and modifying a subgraph of G that contains the same number of nodes as W_i. The resulting graph G^{W_i} is given to user i as the watermarked graph.
- **Watermark extraction**: To identify the watermark in G', we use Ω_i to regenerate W_i and then search for the existence of W_i within G', for each user i.

In this section, we focus on describing the detailed procedure of these two components. We present detailed analysis on the two fundamental properties of graph watermarks, *i.e.* uniqueness and detectability in Section 5.

4.1 Watermark Embedding

The most straightforward way to embed a watermark is to directly attach the watermark graph to the original graph. That is, if W_i represents the watermark graph for user i, and G represents the original graph, the embedding treats W_i as an independent graph, and adds new edges to connect W_i to G. However, this approach has two disadvantages. *First*, direct graph attachment makes it easy for external attackers to identify and remove W_i from G without using graph key K^G and user i's signature $K^i_{priv}(T)$. New edges connecting W_i and G must be carefully chosen to reduce the chance of detection, which is very challenging. *Second*, attaching a (structurally different) subgraph W_i directly to a graph G introduces larger structural distortions.

Instead, we propose an alternative approach that embeds the watermark graph "in-band." That is, the embedding process first selects k nodes (k is the number of nodes in W_i) from G and identifies S, the corresponding subgraph of G induced by these k nodes. It then modifies S using W_i without affecting any other nodes in G. Because the watermark graph W_i is naturally connected with the rest of the graph, both the risk of detection and amount of distortion induced on the original graph G are significantly lower than those of the direct attachment approach.

We now describe the details of "in-band" watermark embedding, which consists of four steps: (1) generating a random generator seed Ω_i from user i's signature $K^i_{priv}(T)$ and graph key K^G; (2) generating the watermark graph W_i from the seed Ω_i; (3) selecting the placement of W_i on G by picking k nodes from G and identifying the corresponding subgraph S induced by these k nodes; and (4) embedding W_i into G by modifying S to match structure of W_i.

Step 1: Generating a random generator seed Ω_i. To generate an unforgettable watermarked graph, we form a random generator seed Ω_i [9] using user i's signature $K^i_{priv}(T)$ and graph key K^G.

Suppose the system intends to generate a watermarked version of graph G at time T to share with user i. We begin by first sending user i with the timestamp T. User i responds with its signature $K^i_{priv}(T)$, by encrypting the timestamp with its private key K^i_{priv}. Before proceeding further, we validate $K^i_{priv}(T)$ to ensure it is from user i, by decrypting it with user i's public key K^i_{pub}. If the timestamps match, we combine the signature $K^i_{priv}(T)$ and the graph key K^G to form the random generator seed Ω_i for user i. A mismatch may indicate that user i is a potential malicious user.

Note that Ω_i cannot be formed alone by the data owner who only holds the graph key K^G, or by user i who only owns its private key K^i_{priv}. Thus, results computed using Ω_i, including the random graph W_i generated (Step 2) and the choice of graph nodes to mark (Step 3), cannot be derived independently by the data owner or identified by user i.

Step 2: Generating the watermark graph W_i. We generate W_i as an Erdos-Renyi random graph with edge probability of p and node count k ($k \ll n$, where n is the number of nodes in G). The random edge generator uses Ω_i as the seed [9]. The k nodes of W_i are ordered as $\{v_1, v_2, ..., v_k\}$.

The key factor here is choosing the node count k and the edge probability p. To ensure watermark uniqueness, Section 5.1 shows that the two parameters must satisfy $k \geq (2+\delta)\log_q n$, where $q = \frac{1}{\max(p,1-p)}$ and δ is a constant > 0. Furthermore, it is easy to prove that $p = \frac{1}{2}$ minimizes the node count k and the average edge count $p \cdot \binom{k}{2}$ of the watermark graph W_i. Intuitively, using a compact watermark graph not only reduces the amount of distortion to G, but also improves its robustness against malicious attacks. Thus, we configure $p = \frac{1}{2}$ and therefore $k = (2+\delta)\log_2 n$. This produces a reasonably sized watermark graph (k <100) even for extremely large graphs, *e.g.* the complete Facebook social graph (∼1 billion nodes in 2014).

Step 3: Selecting the watermark placement on graph G. Next, we identify k nodes from G and its corresponding subgraph S to embed the watermark graph. To ensure reliable extraction, we must choose these k nodes carefully, meeting these two requirements. *First*, using Ω_i generated in Step 1, the k nodes must be chosen deterministically and remain distinguishable from the other nodes of G. *Second*, the set of the k nodes chosen for different watermarks (or different Ω_i values) must be easily distinguishable from each other to reinforce watermark uniqueness. Our biggest challenge in meeting these requirements is that we cannot use node IDs to distinguish nodes from each other. Node IDs or any type of metadata can be easily altered or stripped by attackers before or after leaking G', thereby making extraction impossible.

We address this challenge by using local graph structure around each node as its "label." Specifically, we define a *node structure description* (NSD) as a descriptive feature of each node. A node v's NSD is represented by an array of v's sorted neighbor degrees. For example, if node v has three neighbors with node degrees 2, 6, 4, respectively, then v's NSD label is "2-4-6." We then hash v's NSD label into a numerical value using a secure one-way hash *e.g.* SHA-1 [33], and refer to the result as node v's *NSDhash*.

Next, we use Ω_i as the seed to randomly generate k hash values, and use each as an index (*e.g.* using a mod function) to identify a node in G. It is possible that multiple nodes have the same NSDhash, *i.e.* a collision. If this happens, we resolve the collision by using Ω_i again as an index into a sorted list of these nodes with the same NSDhash. The nodes can be sorted by any deterministic order, *e.g.* node IDs in the original graph. Note that this process is only required for embedding (and not extraction), so any deterministic order chosen by the graph owner will suffice.

At the end of this step, we obtain k ordered nodes from G, $X = \{x_1, x_2, ..., x_k\}$, and the corresponding subgraph $S = G[X]$ induced by the node set X on G.

Step 4: Embedding the watermark graph W_i into graph G. In this step, we embed the watermark graph W_i by modifying the subgraph $S = G[X]$ to match W_i. Specifically, we match each (ranked) node in W_i, $\{v_1, v_2, ..., v_k\}$ with the corresponding node in S (or X), $\{x_1, x_2, ..., x_k\}$, *i.e.* $f : W \to S, f(v_i) = x_i$. And once the nodes are mapped, we then apply an XOR operation on each edge of the two graphs. That is, we consider the connection between (v_i, v_j) or (x_i, x_j) as one bit, *i.e.* an edge between (v_i, v_j) or (x_i, x_j) means 1 and no edge between (v_i, v_j) or (x_i, x_j) means 0. If an edge (v_i, v_j) exists in W_i, we modify the corresponding edge value in S from (x_i, x_j) to $(x_i, x_j) \oplus 1$; and if no edge (v_i, v_j) exists in W_i, we modify the edge value (x_i, x_j) to $(x_i, x_j) \oplus 0$. When the above edge modification process ends, we also explicitly create edges between nodes x_i and x_{i+1} to maintain a connected subgraph. As a result, we transfer the subgraph S into S^{W_i} with the watermark graph W_i embedded. The reason for choosing the XOR operation is that it allows the same watermark to be embedded in the graph multiple times (at multiple locations), thus reducing the risk of the watermark being detected and destroyed by attacks such as frequent subgraph mining. We will discuss this in more details in Section 6.

At the end of this step, we obtain a watermarked graph G^{W_i} for user i. Before we distribute it to user i, we anonymize G^{W_i} by completely (randomly) reassigning all node IDs. Such anonymization not only helps to protect user privacy, but also minimizes the opportunity for colluding attackers with multiple watermarked graphs to identify the embedded watermark (see Section 6).

4.2 Watermark Extraction

The watermark *extraction* process determines if a watermark graph W_i is embedded in a target graph G'. If so, then G' is a legitimate copy distributed to user i. The extraction process faces two key challenges. *First*, the target graph G' can easily be modified by users/attackers during the graph distribution process. In particular, all node IDs can be very different from that of the original G. Thus extraction cannot rely on node IDs in G'. *Second*, identifying whether a subgraph exists in a large graph is equivalent to a subgraph matching problem, known to be NP-complete. To handle large graphs, we need a computationally efficient algorithm.

Our design addresses these two challenges by leveraging knowledge on the structure of the subgraph where the watermark was embedded. This eliminates the dependency on node IDs while significantly reducing the search space during the subgraph matching process. We describe our proposed design in detail below.

Step 1: Regenerating the watermark. The owner performs the extraction, and has access to the original graph G, graph key K^G, and user's signature $K^i_{priv}(T)$. For each user i, we combine its signature $K^i_{priv}(T)$ and graph key K^G to form its random generator seed Ω_i. Then, we follow step $2-4$ described in Section 4.1 to regenerate the watermark graph W_i, identify the k ordered nodes from G and their NSD labels, and finally the modified subgraph S^{W_i} that was placed on a "clean" version of the watermarked graph G^{W_i}.

Step 2: Identifying candidate watermark nodes on G'. Given the k nodes $X = \{x_1, x_2, ..., x_k\}$ identified from the original graph G, in this step we need to identify for each x_j, a set of candidate nodes on the target graph G' that can potentially become x_j. We accomplish this by identifying all the nodes on G' whose NSD labels are the same of x_j in the "clean" version of the watermarked graph G^{W_i}. Since multiple nodes can have the same NSD label, this process will very likely produce multiple candidates. To shrink the candidate list, we examine the connectivity between candidate nodes of X on G' and compare it to that among X on G^{W_i}. If two nodes x_m and x_n are connected in G^{W_i}, we prune their candidate node lists by removing any candidate node of x_m that has no edge with any candidate node of x_n on G' and vice versa. This pruning process dramatically reduces the search space. After this step, we obtain for each x_i the candidate node list C_i on the target graph G'.

Step 3: Detecting watermark graph S^{W_i} on G'. Given the candidate node list of each node in X, we now search for the existence of S^{W_i} on the target graph G'. For this we apply a recursive algorithm to enumerate and prune the combinations of the candidate sets, until we identify S^{W_i} or exhaust all the node candidates. The detailed algorithm is listed in Algorithm 1. In this

Algorithm 1 Recursive Algorithm for Detecting S^{W_i} on G'.

Function: SubgraphDetection(G', S^{W_i}, $\{C_1, C_2, ..., C_k\}$, Y, m)
Input: Graph G', watermark graph S^{W_i}, candidate node list C_i for each node x_i in X, identified node list $Y = \{y_1, y_2, ..., y_m\}$ $(m < k)$
Output: Identified node list $Y = \{y_1, y_2, ..., y_{m+1}\}$
for each node $c \in C_{m+1}$ **do**
  if $c \notin Y$ and each edge (c, y_t) in G' $(t = 1..m)$ is the same as the edge (x_{m+1}, x_t) in S^{W_i} $(t = 1..m)$ **then**
    $Y = Y \cup c$
    $m = m + 1$
    if $m == k$ **then**
      Return Y
    else
      SubgraphDetection(G', S^{W_i}, $\{C_1, C_2, ..., C_k\}$,Y, m)
    end if
    $Y = Y \setminus c$
    $m = m - 1$
  end if
end for
Return Y

algorithm, we use a node list Y to record the nodes in G' which we have already finalized as the corresponding nodes in S^{W_i}, *i.e.* $Y = \{y_1, y_2, ..., y_m\}$ $(m \leq k)$. When the process starts, $Y = \emptyset$, $m = 0$.

Discussion. The above design shows that our watermark extraction algorithm simplifies the subgraph search problem by restricting it to a small number of selected nodes from a graph, thus avoiding the NP-complete subgraph matching problem.

To illustrate the efficiency of our algorithm, we now show an estimation of the computational complexity. Assume that the number of candidates for each watermark node x_i is $|C_i|$, and the probability that an edge between node $c_{im} \in |C_i|$ and node $c_{jn} \in |C_j|$ is p_{ij}. Moreover, since we prove that the probability of an edge between node x_i and node x_j is $\frac{1}{2}$ in Section 5.1, the probability that the connectivity between (c_{im}, c_{jn}) matches the connectivity between (x_i, x_j) is $\frac{1}{2} \cdot p_{ij} + \frac{1}{2} \cdot (1 - p_{ij}) = \frac{1}{2}$. We can show that to identify a node list with m nodes in Algorithm 1, we need to match $\binom{m}{2}$ node pairs. Thus, the probability to identify a node list with m nodes is $\frac{1}{2}^{\binom{m}{2}}$, and the expected number of node combinations is $\prod_{i=1}^{m} |C_i| \cdot \frac{1}{2}^{\binom{m}{2}}$. Thus, the computational complexity of Algorithm 1 is proportional to the sum of node combinations at each step, *i.e.* $O(\sum_{m=2}^{k} \prod_{i=1}^{m} |C_i| \cdot \frac{1}{2}^{\binom{m}{2}})$. Note that we do not consider the fixed $k - 1$ edges between (x_{i-1}, x_i) for simplicity.

This result shows that as more nodes are identified in Algorithm 1, fewer node combinations exists, which approximates to 0 (as shown in Section 5.1). This means the major computation cost of our algorithm comes from the initial few steps and is dominated by the size of their candidates. Note that we target real graphs with very high level of node heterogeneity, *e.g.* small-world, power-law or highly clustered graphs, which leads to small candidate size in most cases. In other words, the computational complexity of our algorithm is low in real graphs. In practice, our system can efficiently extract watermarks from real, million-node graphs, and do so in a few minutes on a single commodity server (Section 7.3).

5. FUNDAMENTAL PROPERTIES

Having described the basic watermark system, we now present detailed analysis on its two fundamental properties: *watermark uniqueness* where each watermark must be unique to the corresponding user, and *watermark detectability* where the presence of a watermark should not be easily detectable by external users without the knowledge of the seed Ω_i associated with user i.

5.1 Watermark Uniqueness

As a proof of ownership, each embedded watermark should be unique for its user. That is, given the original graph G and the seed Ω_i associated with user i, the embedded watermark graph S^{W_i} should not be isomorphic to any subgraph of G^{W_j} $(i \neq j)$ where G^{W_j} is the watermarked graph for user j. Meanwhile, S^{W_i} should not be isomorphic to any subgraph of the original graph G. The following proof shows that with high probability, our proposed graph watermark system produces unique watermarks for any graph G.

THEOREM 1. *Given a graph G with n nodes, let $k \geq (2 + \delta) \log_2 n$ for a constant $\delta > 0$. We apply the following process to create a watermarked graph G^{W_i} for user i:*

- *We create k nodes, $V = \{v_1, v_2, ..., v_k\}$, and generate a random graph W_i on V with an edge probability of $\frac{1}{2}$.*
- *We randomly select k nodes, $X = \{x_1, x_2, ..., x_k\}$ from G, and identify the subgraph corresponding to these k nodes $S = G[X]$.*
- *Using W_i, we modify S as follows: we first map each node x_i in X to a node v_i in V. Let $e(u, v) = 1$ denote an edge exists between node u and v and $e(u, v) = 0$ denote otherwise. We modify each $e(x_i, x_j)$ in S to $e(x_i, x_j) \oplus e(v_i, v_j)$. We then explicitly connect nodes x_i and x_{i+1},* i.e. *$e(x_i, x_{i+1}) = 1$. The resulting S now becomes S^{W_i}, and the resulting G becomes G^{W_i}.*

Let G^{W_l} denote a watermarked graph for user l $(l \neq i)$, built using a different seed Ω_l. Then with low probability, any subgraph of G^{W_l} or G is isomorphic to S^{W_i}.

PROOF. We first show that with low probability, any subgraph of G^{W_l} is isomorphic to S^{W_i}. Let $Y = \{y_1, y_2, ...y_k\}$ be a set of ordered nodes in G^{W_l}, where each y_i maps to a node x_i in X. We define an event $\mathcal{E}_Y$ occurs if the subgraph $G^{W_l}[Y]$ is isomorphic to $G^{W_i}[X]$ or S^{W_i}. Then the event $\mathcal{E}$ representing the fact that there exists at least one subgraph on G^{W_l} that is isomorphic to S^{W_i} is the union of events $\mathcal{E}_Y$ on all possible Y, *i.e.* $\mathcal{E} = \cup_Y \mathcal{E}_Y$.

Next, we compute the probability of event $\mathcal{E}$ by those of individual event $\mathcal{E}_Y$. Specifically, we first show that the probability of an edge exists between node x_i and x_j $(j \neq i + 1)$ in $S^{W_i} = G^{W_i}[X]$ is $\frac{1}{2}$. This is because each edge in the random graph W_i is independently generated with probability $\frac{1}{2}$. After performing the XOR operation between W_i and S, the probability of an edge exists between x_i and x_j $(j \neq i + 1)$ on S^{w_i} is $\frac{1}{2} \cdot p_{ij} + (1 - p_{ij}) \cdot \frac{1}{2} = \frac{1}{2}$ where p_{ij} is the probability that an edge exists between x_i and x_j on S. Thus the result of XOR between W_i and S is also a random graph, and its edge generation is independent of that in $G^{W_l}, l \neq i$. Furthermore, it is easy to show that our design applies XOR operations on $\binom{k}{2} - (k - 1)$ node pairs on the k nodes, and each node pair has an edge with a probability of $\frac{1}{2}$. Thus, the probability of a subgraph $G^{W_l}[Y]$ being isomorphic to S^{W_i} is $P(\mathcal{E}_Y) = \frac{1}{2}^{\binom{k}{2}-(k-1)} \cdot \beta$ where $\beta \leq 1$ is the probability that every (y_i, y_{i+1}) pair in $G^{W_l}[Y]$ is connected. Thus $P(\mathcal{E}_Y) \leq \frac{1}{2}^{\binom{k}{2}-(k-1)}$.

Since $\mathcal{E} = \cup_Y \mathcal{E}_Y$ and there are less than n^k possible sets of k ordered nodes in G^{W_l}, we use the Union Bound to compute the probability of event $\mathcal{E}$ as follows:

$$P(\mathcal{E}) < n^k \cdot P(\mathcal{E}_Y) \leq n^k \cdot \frac{1}{2}^{\binom{k}{2}-(k-1)} = 2^{\frac{k^2}{2+\delta}} \cdot \frac{1}{2}^{\frac{k^2-3k}{2}+1} = \frac{1}{2}^{\frac{\delta k^2}{2(2+\delta)} - \frac{3k}{2} + 1} \quad (1)$$

The above equation shows that the probability $P(\mathcal{E})$ reduces exponentially to 0 as k increases.

Graph Category	Graph	# of Nodes	# of Edges	Avg. Deg.	k	Node Degree Criterion		k-node Subgraph Density Criterion		Suitability
						$(k+1)/2$	$[N_{min}(G), N_{max}(G)]$	Watermark	$[D_{min}(k), D_{max}(k)]$	
Facebook	Russia	97,134	289,324	6.0	39	20	[1, 748]	390	[45, 701]	**Yes**
	L.A.	603,834	7,676,486	25.4	45	23	[1, 2141]	517	[44, 975]	**Yes**
	London	1,690,053	23,084,859	27.3	48	24	[1, 1483]	588	[47, 1128]	**Yes**
Other Social Networks	Epinions (1)	75,879	405,740	10.7	38	19	[1,3044]	370	[47,649]	**Yes**
	Slashdot (08/11/06)	77,360	507,833	13.1	38	19	[1, 2540]	370	[38, 668]	**Yes**
	Twitter	81,306	1,342,303	33.0	38	19	[1, 3383]	370	[44, 703]	**Yes**
	Slashdot (09/02/16)	81,867	497,672	12.2	38	19	[1, 2546]	370	[38, 669]	**Yes**
	Slashdot (09/02/21)	82,140	500,481	12.2	38	19	[1, 2548]	370	[38, 669]	**Yes**
	Slashdot (09/02/22)	82,168	543,381	13.2	38	19	[1, 2553]	370	[38, 673]	**Yes**
	GPlus	107,614	12,238,285	227.5	39	20	[1, 20127]	389.5	[53, 741]	**Yes**
	Epinions (2)	131,828	711,496	10.8	40	20	[1, 3558]	409.5	[51, 780]	**Yes**
	Youtube	1,134,890	2,987,624	5.3	47	24	[1, 28754]	563.5	[47, 815]	**Yes**
	Pokec	1,632,803	22,301,964	27.3	48	24	[1, 14854]	587.5	[47, 979]	**Yes**
	Flickr	1,715,255	15,555,041	18.1	48	24	[1, 27236]	588	[51, 1128]	**Yes**
	Livejournal	5,204,176	48,942,196	18.8	52	26	[1, 15017]	689	[51, 1326]	**Yes**
Citation Networks	Patents	23,133	93,468	8.1	34	17	[1, 280]	297	[37, 373]	**Yes**
	ArXiv (Theo. Cit.)	27,770	352,304	25.4	34	17	[1, 2468]	297	[36, 534]	**Yes**
	ArXiv (Phy. Cit.)	34,546	420,899	24.4	35	18	[1, 846]	314.5	[36, 544]	**Yes**
Collaboration Networks	ArXiv (Phy.)	12,008	118,505	19.7	32	16	[1, 491]	263.5	[45, 496]	**Yes**
	ArXiv (Astro)	18,772	198,080	21.1	33	17	[1, 504]	280	[37, 528]	**Yes**
	DBLP	317,080	1,049,866	6.6	43	22	[1,343]	472.5	[43,903]	**Yes**
	ArXiv (Condense)	3,774,768	16,518,947	8.8	51	26	[1, 793]	663	[50,1063]	**Yes**
Communication Networks	Email (Enron)	36,692	183,831	10.0	35	18	[1,1383]	314.5	[43,515]	**Yes**
	Email (Europe)	265,214	365,025	2.8	42	21	[1,7636]	451	[74,683]	**Yes**
	Wiki	2,394,385	4,659,565	3.9	49	25	[1, 100029]	612	[65, 1066]	**Yes**
Web graphs	Stanford	281,903	1,992,636	14.1	42	21	[1,38625]	451	[66,861]	**Yes**
	NotreDame	325,729	1,103,835	6.8	43	22	[1,10721]	472.5	[60,903]	**Yes**
	BerkStan	685,230	6,649,470	19.4	45	23	[1,84230]	517	[79,990]	**Yes**
	Google	875,713	4,322,051	9.9	46	23	[1, 6332]	540	[72, 1033]	**Yes**
Location based OSNs	Brightkite	58,228	214,078	7.4	37	19	[1,1134]	351	[41,665]	**Yes**
	Gowalla	196,591	950,327	9.7	41	21	[1,14730]	430	[44,723]	**Yes**
AS Graphs	Oregon (1)	11,174	23,409	4.2	31	16	[1,2389]	247.5	[95,352]	**Yes**
	Oregon(2)	11,461	32,730	5.7	32	16	[1,2432]	263.5	[79,476]	**Yes**
	CAIDA	26,475	53,381	4.0	34	17	[1,2628]	297	[113,436]	**Yes**
	Skitter	1,696,415	11,095,298	13.1	48	24	[1, 35455]	588	[52, 1128]	**Yes**
P2P networks	Gnutella (02/08/04)	10,876	39,994	7.4	31	16	[1,103]	247.5	[30,80]	No
	Gnutella (02/08/25)	22,687	54,705	4.8	34	17	[1,66]	297	[0,0]	No
	Gnutella (02/08/24)	26,518	65,369	4.9	34	17	[1,355]	297	[0,44]	No
	Gnutella (02/08/30)	36,682	88,328	4.8	35	18	[1,55]	314.5	[35,70]	No
	Gnutella (02/08/31)	62,586	147,892	4.7	37	19	[1, 95]	351	[39,76]	No
Amazon Co-purchasing Networks	Amazon (03/03/02)	262,111	899,792	6.9	42	21	[1,420]	451	[88,132]	No
	Amazon (2012)	334,863	925,872	5.5	43	22	[1,549]	472.5	[0,0]	No
	Amazon (03/03/12)	400,727	2,349,869	11.7	43	22	[1,2747]	472.5	[52,285]	No
	Amazon (03/06/01)	403,394	2,443,408	12.1	43	22	[1, 2752]	473	[52, 333]	No
	Amazon (03/05/05)	410,236	2,439,437	11.9	43	22	[1,2760]	472.5	[50,333]	No
Road Networks	Pennsylvania	1,088,092	1,541,898	2.8	47	24	[1,9]	563.5	[0,0]	No
	Texas	1,379,917	1,921,660	2.8	47	24	[1,12]	563.5	[0,0]	No
	California	1,965,206	2,766,607	2.8	49	25	[1, 12]	612	[0, 0]	No

Table 1: Suitability of watermarking for 48 of today's network graphs, determined by comparing their node degree distribution $[N_{min}(G), N_{max}(G)]$ and k-node subgraph density $[D_{min}(k), D_{max}(k)]$ to those of the embedded watermark graphs. 35 out of these 48 graphs are suitable for watermarking.

Finally, we can apply the same method to show that with low probability, any subgraph of G is isomorphic to S^{W_i}. This is because the XOR operations between W_i and S produce a random graph that is independent of G. □

5.2 Watermark Detectability

In addition to providing uniqueness, a practical watermark design should also offer low detectability, *i.e.*, with low probability each watermark gets identified by external users/attackers. This means that without knowing the seed Ω_i associated with user i, the embedded watermark graph S^{W_i} should not be easily distinguishable from the rest of the graph G^{W_i}. Therefore, the detectability would depend heavily on the topology of the original graph G, *i.e.* a watermark graph can be well hidden inside a graph G^{W_i} if its structural property is not too different from that of G.

In the following, we examine the detectability of watermarks in terms of *a graph's suitability for watermarking*. This is because directly quantifying the detectability is not only highly computational expensive[1], but also lacks a proper metric. Instead, we cross-compare the key structural properties of S^{W_i} and G, and define G as being suitable for watermarking if its structure properties are similar to that of S^{W_i}, implying a low watermark detectability.

Suitability for Watermarking. To evaluate a graph's suitability for watermarks, we first study the key structural property of the embedded watermark graph S^{W_i}. To guarantee watermark uniqueness and minimize distortion, the watermark graph S^{W_i} needs to be a random graph with an edge probability of $\frac{1}{2}$ (except for the fixed edges between x_i, x_{i+1} node pairs), and include $k = (2+\delta)\log_2 n$ nodes. Thus its average node degree is at least $(k+1)/2$ and its average graph density is $(\binom{k}{2} + k - 1)/2$.

[1]Each embedded watermark graph is similar to a random graph with $\frac{1}{2}$ edge probability. Thus the detectability is low if certain subgraphs of G are also random graphs with similar edge probabilities. Yet identifying these subgraphs (and the embedded watermark graph) on a large graph incurs significant computation overhead.

Graph	Subgraph		Watermark Graph		Suitability
	Node #	Avg. Deg.	k	Avg. Deg.	
Russia	4,794	22.2	39	20.0	**Yes**
L.A.	196,174	49.2	45	23.0	**Yes**
London	562,075	56.1	48	24.5	**Yes**
Epinions (1)	7,083	68.7	38	19.5	**Yes**
Slashdot (08/11/06)	9,908	53.4	38	19.5	**Yes**
Twitter	34,014	60.5	38	19.5	**Yes**
Slashdot (09/02/16)	10,065	53.0	38	19.5	**Yes**
Slashdot (09/02/21)	10,105	53.2	38	19.5	**Yes**
Slashdot (09/02/22)	10,605	53.4	38	19.5	**Yes**
GPlus	68,828	347.1	39	20.0	**Yes**
Epinions (2)	10,363	83.5	40	20.5	**Yes**
Youtube	31,720	45.1	47	24.0	**Yes**
Pokec	564,001	53.0	48	24.5	**Yes**
Flickr	136,202	174.5	48	24.5	**Yes**
Livejournal	945,567	57.5	52	26.5	**Yes**
Patents	2,370	15.6	34	17.5	**Yes**
ArXiv (Theo. Cit.)	12,054	43.4	34	17.5	**Yes**
ArXiv (Phy. Cit.)	14,785	37.9	35	18.0	**Yes**
ArXiv (Phy.)	2,860	62.5	32	16.5	**Yes**
ArXiv (Astro)	6,536	42.9	33	17.0	**Yes**
DBLP	15,004	17.3	43	22.0	**Yes**
ArXiv (Condense)	178,455	16.0	51	26.0	**Yes**
Email (Enron)	3,481	48.2	35	18.0	**Yes**
Email (Europe)	1,779	44.0	42	21.5	**Yes**
Wiki Talk	21,253	83.1	49	25.0	**Yes**
Stanford	35,600	42.1	42	21.5	**Yes**
NotreDame	16,831	38.7	43	22.0	**Yes**
BerkStan	110,202	57.0	45	23.0	**Yes**
Google	55,431	14.8	46	23.5	**Yes**
Brightkite	4,586	30.8	37	19.0	**Yes**
Gowalla	17,946	39.3	41	21.0	**Yes**
Oregon (1)	264	17.1	31	16.0	**Yes**
Oregon(2)	579	31.0	32	16.5	**Yes**
CAIDA	575	16.0	34	17.5	**Yes**
Skitter	146,601	50.0	48	24.5	**Yes**
Gnutella (02/08/04)	796	5.2	31	16.0	No
Gnutella (02/08/25)	499	2.0	34	17.5	No
Gnutella (02/08/24)	709	2.7	34	17.5	No
Gnutella (02/08/30)	1,001	3.8	35	18.0	No
Gnutella (02/08/31)	1,276	3.6	37	19.0	No
Amazon (03/03/02)	3,727	2.8	42	21.5	No
Amazon (2012)	5,318	2.5	43	22.0	No
Amazon (03/03/12)	25,717	6.7	43	22.0	No
Amazon (03/06/01)	28,081	7.3	43	22.0	No
Amazon (03/05/05)	28,044	7.5	43	22.0	No
Pennsylvania	0	0	47	24.0	No
Texas	0	0	47	24.0	No
California	0	0	49	25.0	No

Table 2: Size and density of subgraph on nodes with degree $> (k+1)/2$ in each graph. Size is the number of subgraph nodes, and density is quantified as average edges each node having inside the subgraph.

Given these properties of the embedded watermark, we note that watermark node degree and density can be higher than those of many real-world graphs, such as those listed in Table 1. Intuitively, to ensure low detectability of such a watermark graph, suitable graphs should include a set of nodes (D) that are difficult to distinguish from the watermark nodes in term of node degree and subgraph density. Specifically, a suitable graph dataset needs to contain a set of nodes D with degree comparable or higher than the watermark graph node degree; and the density of the subgraph on D is at least comparable to the watermark graph density. If these two properties hold, the embedded watermark graph cannot be easily distinguished from D in the graph, and therefore cannot be detected by attackers.

To capture the above intuition, we define that a graph G is suitable for watermarking if its node degree and graph density satisfy the following two criteria. First, the minimum and maximum node degree of G, denoted as $N_{min}(G)$ and $N_{max}(G)$ respectively, need to satisfy $N_{min}(G) \leq (k+1)/2 \leq N_{max}(G)$. Second, across all k-node subgraphs of G whose node degree expectation is greater than $(k+1)/2$, the minimum and maximum graph density need to satisfy $D_{min}(k) \leq (\binom{k}{2} + k - 1)/2 \leq D_{max}(k)$ [2]. Together, these two criteria ensure that the embedded watermark graph can be "well hidden" inside G^{W_i}.

Suitability of Real Graph Datasets. We measure the suitability of watermarks in 48 real networks graphs. These graphs represent vastly different types of networks and a wide range of structural topologies with size ranging from $10K$ nodes and $39K$ edges to $5M$ nodes and $48M$ edges. These graphs represent vastly different types of networks and a wide range of structural topologies. They include 3 social graphs generated from Facebook regional networks matching Russia, L.A., and London [40]. They include 12 other graphs from online social networks, including Twitter [21], Youtube [43], Google+ [21], Slovakia Pokec [38], Flickr [26], Livejournal [26], 2 snapshots from Epinions [32], and 4 snapshots from Slashdot [20]. We also add 3 citation graphs from arXiv and U.S. Patents [18], 4 graphs capturing collaborations in arXiv [18] and DBLP [43], 3 communication graphs generated from 2 Email networks [19, 20] and Wiki Talk [17], 4 web graphs [16, 2], 2 location-based online social graphs from Brightkite and Gowalla [6], 5 snapshots of P2P file sharing graph from Gnutella [19], 4 Internet Autonomous System (AS) maps [18], 5 snapshots of Amazon co-purchasing networks [15, 43], and 3 U.S. road graphs [16]. The statistics of all graphs are listed in Table 1.

For all graphs, we use $\delta = 0.3$ to ensure a 99.999% watermark uniqueness, and list their watermark size k in Table 1. We also show the two above criteria: node degree and k-node subgraph density. If a graph satisfies both criteria, our results will hold for any watermarks embedded on it.

We can make two observations from Table 1. *First*, 35 out of our 48 total graphs are suitable for watermarking. Also note that graphs describing similar networks are consistent in their suitability. For example, all 15 graphs from various online social networks are suitable for watermarks! *Second*, all the 13 graphs unsuitable for watermarks come from only 3 kinds of networks, *i.e.* copurchasing networks, P2P networks, and Road networks. These results in each group are self consistent. These results support our assertion that our proposed watermarking mechanism is applicable to most of today's network graphs with low detection risk. In practice, the owner of a graph can apply the same mechanism to determine if her graph is suitable for our watermark scheme.

To understand key properties determining whether a graph is suitable for watermarking, we measure various graph structural properties, including average node degree, node degree distribution, clustering coefficient, average path length, and assortativity. We also consider the size and density of subgraphs on nodes with degree more than watermark minimum average degree $(k+1)/2$. Our measurement results show that the size and density of subgraphs on nodes with degree $> (k+1)/2$ are the most important properties to determine suitability. Here, the size of these subgraphs is the number of nodes in the subgraph, and the density of the subgraph is measured as the average edges each node has inside the subgraph, *i.e.* average degree inside the subgraph. As shown in Table 2, unsuitable graphs do not have subgraphs with density to comparable to watermarks, while subgraphs with the desired den-

[2]To avoid computationally prohibitive subgraph enumeration, we apply a sampling method to estimate them with full details in [45].

sity can be found in graphs deemed suitable. These results are consistent with our intuition on quantifying suitability of watermarks.

Summary. Since the average watermark subgraph has high node degree and density, a graph suitable for watermarking must include a set of nodes, whose degree and subgraph density are comparable or even higher than watermark subgraphs. We propose two criteria targeting at node degree and subgraph density respectively to quantify whether a graph is suitable for watermarking. We collect a large set of available graph datasets and find 35 out of 48 real graphs are suitable. We expect similar suitability results in other real network graphs.

6. MORE ROBUST WATERMARKS

Our basic design provides the fundamental building blocks of graph watermarking with little consideration of external attacks. In practice, malicious users can seek to detect or destroy watermarked graphs. Here, we first describe external attacks on watermarks, and then present advanced features that defend against the attacks. Note that these improvement techniques aim to increase the cost of attacks rather than disabling them completely. Finally, we re-evaluate the watermark uniqueness of the advanced design.

6.1 Attacks on Watermarks

As discussed earlier, our attack model includes attacks trying to destroy watermarks while preserving the topology of the original graph. Based on the number of attackers, attacks on watermarks fall under our two models: single attacker and colluding attackers. With access to only one watermarked graph, a single attacker can modify nodes and/or edges in the graph to destroy watermarks. With multiple watermarked graphs, colluding attackers can perform more sophisticated attacks by cross-comparing these graphs to detect or remove watermarks.

Single Attacker Model. The naive edge attack is easiest to launch, and tries to disrupt the watermark by randomly adding or removing edges on the watermarked graph. For the attacker, there is a clear tradeoff between the severity of the attack (number of edges or nodes modified), and the structural change or distortion applied to the graph structure.

At first glance, this attack seems weak and unlikely to be a real threat. The probability of the attacker modifying one edge or node in the embedded watermark graph W_i is extremely low, given the relatively small size of W_i compared to the graph. As shown later, however, this attack can be quite disruptive in practice. By modifying a node n_i or an edge connected to n_i, the attack impacts all of n_i's neighboring nodes, since their NSD labels will be modified. These NSD label changes, while small, are enough to make locating nodes in the watermark graph very difficult. This effect is exacerbated in social graphs that exhibit a small world structure, since any change to a supernode's degree will impact a disproportionately large portion of nodes in the graph.

Some versions of this attack would either release a partial subgraph of the watermarked graph, or merge multiple watermarked graphs. In both cases, this destroys the embedded watermarks, but also significantly distorts the graphs and reduces their usability. We do not consider these disruptive attacks in our study, and target them for future work.

Collusion Attacks. By obtaining multiple watermarked graphs, an attacker can compare these graphs to eliminate watermarks. Since we anonymize each watermarked graph by randomly reassigning node IDs (see Section 4.1), attackers cannot directly match individual nodes across graphs. To compare multiple graphs, we apply the deanonymization methods proposed in [27, 28]. Specifically, we first match 1000 highest degree nodes between two graphs based on their degree and neighborhood connectivity [28], and then start from these nodes to find new mappings with the network structure and the previously mapped nodes [27].

Using deanonymization techniques, attackers can then build a "clean" graph, where an edge exists if it exists in the majority of the watermarked graphs. Since embedded watermark graphs are likely embedded at different locations on each graph, a majority vote approach effectively removes the contributions from watermark subgraphs, leading to a graph that closely approximates the original G.

6.2 Improving Robustness against Attacks

The attacks discussed above can disrupt the watermark extraction process in two ways. First, adding or deleting nodes/edges in G' changes node degrees, and therefore nodes' NSD labels, thereby disrupting the identification of candidate nodes during the second step of the extraction process; second, adding or deleting nodes/edges inside the embedded watermark graph S^{W_i} can change the structure of the watermark graph, making it difficult to identify during the third step of the extraction process. To defend against these attacks, we propose five improvements over the basic extraction design to produce an improved watermark generation algorithm.

Improvements #1, #2: Addressing changes to node neighborhoods. Extracting a watermark involves searching through nodes in G' by their NSD labels. By adding or deleting nodes/edges, attackers can effectively change NSD labels across the graph. To address this, we propose two changes to the basic extraction design. *First*, we bucketize node degrees (with bucket size B) to reduce the sensitivity of a node's NSD label to its neighbors' node degrees. For example, with $B = 5$, a node with degree 9 will stay in the same bucket even if one of its edges has been removed (reducing its node degree to 8). *Second*, when selecting a watermark node's candidate node list, we replace the exact NSD label matching with the approximate NSD label matching. A match is found if the overlap between two bucketized NSD labels exceeds a threshold θ. For example, with $\theta = 50\%$, a node with bucketized NSD label "1-2-3-4" would match a node with label "1-2-3" since the overlap is $75\% > \theta$.

These changes clearly allow us to identify more candidates for each watermark node, thus improving robustness against small local modifications. On the other hand, more candidates lead to more computation during the subgraph matching step (step 3 in Section 4.2). Such expansion, however, does not affect watermark uniqueness and detectability, since they are unrelated to the size of candidate pools.

Improvement #3, #4: Addressing changes to subgraph structure. Random changes made to G' may directly impact a node or edge in the embedded watermark. To address this, we propose two techniques. *First*, we add redundancy to watermarks by embedding the same watermark graph W_i into m disjoint subgraphs $S_1, S_2, ...S_m$ from the original graph G. This greatly increases the probability of the owner locating at least one unmodified copy of W_i during extraction, even in the presence of attacks that make significant changes to nodes and edges in G'. Note that since we embed watermarks on disjoint subgraphs, this does not affect watermark uniqueness $1 - P(\mathcal{E})$. While embedding m watermarks will impact false positive, which is $1 - (1 - P(\mathcal{E}))^m$.

Second, it is still possible that all the watermark graphs are "destroyed" by the attacker and there are no matches in the extraction process. If this happens, we replace the exact subgraph matching in the step 3 of the extraction process with the approximate sub-

graph matching. That is, a subgraph matches the watermark graph if the amount of edge difference between the two is less than a threshold L. By relaxing the search criteria used in step 3 of the extraction process, this technique allows us to identify "partially" damaged watermarks, thus again improving robustness against attacks. However, it can also increase false positives in watermark extraction, reducing watermark uniqueness. We show in Section 6.3 that the impact on watermark uniqueness can be tightly bounded by controlling L.

Improvement #5: Addressing Collusion Attacks. Recall that for powerful attackers able to match graphs at an individual node level, they can leverage majority votes across multiple watermarked graphs to remove watermarks. To defend against this, our insight is to embed watermarks that have some portion of spatial overlap in the graph, such that those components will survive majority votes over graphs.

We propose a *hierarchical* watermark embedding process to defend against collusion attacks. To build watermarked graphs for M users, we uniform-randomly divide the M users into 2 groups (a_1 and a_2) and associate each group with a public-private key pair $< K^{a_1}_{pub}, K^{a_1}_{priv} >$ or $< K^{a_2}_{pub}, K^{a_2}_{priv} >$, which is generated and held by the data owner. We repeat this to randomly divide M users into another 2 groups (b_1 and b_2) associated with group key pairs $< K^{b_1}_{pub}, K^{b_1}_{priv} >$ and $< K^{b_2}_{pub}, K^{b_2}_{priv} >$ separately. After this step, each user is assigned to two groups [3]. For example, a user i is assigned to groups a_1 and b_2.

For user i, we then follow step 2-4 in Section 4.1 to embed the two *group watermarks* and its *individual watermark*. Specifically, by receiving user i's signature $K^i_{priv}(T)$, we first generate three seeds: Ω_i by combining $K^i_{priv}(T)$ and K^G, Ω_{a_1} by combining $K^{a_1}_{priv}$ and K^G, and Ω_{b_2} by combining $K^{b_2}_{priv}$ and K^G, where K^G is graph key for graph G. With the two group seeds Ω_{a_1} and Ω_{b_2}, we generate and embed two non-overlap group watermarks. Then we use user i's individual seed Ω_i to embed an individual watermark without overlapping with either of the embedded group watermarks. Note that because the group and individual watermarks are generated with different seeds, this hierarchical embedding process does not affect watermark uniqueness.

Under this design, a collusion attack can successfully destroy all the watermarks only if the attacker can perfectly match each individual node, and the majority of the graphs come from different groups. Otherwise, the majority vote on raw edges will preserve the *group watermark*. We can compute the upper bound of the attack success rate by Equation 2, *i.e.* the probability that the majority of the graphs obtained by the attacker come from different groups:

$$\lambda(M_a, J) = \left(1 - J \sum_{i=\lceil \frac{M_a+1}{2} \rceil}^{M_a} \binom{M_a}{i} \cdot (\frac{1}{J})^i \cdot (\frac{J-1}{J})^{M_a - i}\right)^2 \quad (2)$$

where M_a is the number of watermarked graphs obtained by the attacker and J is the number of groups in each group partition. The above design chose $J = 2$ because it minimizes $\lambda(M_a, J), \forall M_a$. Furthermore, when M_a is odd, $\lambda(M_a, 2) = 0$; and when M_a is even, $\lambda(M_a, 2)$ is at most 0.25 when $M_a = 2$. Note that in equation (2) the operation $(.)^2$ is due to the fact that we group the users twice into two different group classes: a_1, a_2 and b_1, b_2. If we only perform the group partition once (*e.g.* dividing the users into a_1, a_2), then $\lambda(2, 2) = 0.5$. In practice we can further reduce λ by performing multiple rounds of group division (2 in the above design) and adding more group watermarks.

[3] More details about the group assignment are in [45].

Note that group watermarks contain much less information than single user watermarks. In fact, the more robust a group watermark, the larger granularity (and less precision) it will provide. Our proposed solution is to extend the system by using additional "dimensions," *e.g.* go beyond the two dimensions of a and b mentioned above. Combining results from multiple dimensions will quickly narrow down the set of potential users responsible for the leak. However, since a colluding attack requires the involvement of multiple leakers, even identifying a single leaker is insufficient. Developing a scheme to reliably detect multiple (ideally all) colluding users is a topic for future work.

6.3 Impact on Watermark Uniqueness

To improve the robustness of our watermark system, we relax the subgraph matching criteria from exact matching to approximate matching with at most L edge difference. Such relaxation does not affect watermark detectability because it does not change the embedding process. However, it may affect watermark uniqueness, which we will analyze next.

Consider two watermarked graphs G^{W_i} and G^{W_j} that were independently generated for user i and j following the three steps defined in Theorem 1. Let S^{W_i} and S^{W_j} represent the embedded watermark graph in G^{W_i} and G^{W_j}, respectively. To examine the watermark uniqueness, we seek to compute the probability that a subgraph in G^{W_j} differs from S^{W_i} by at most L edges.

Our analysis is similar to Theorem 1's proof. Let $\mathcal{E}_Y$ denote the event where a subgraph of G^{W_j} built on k nodes $Y = \{y_1, y_2, ..., y_k\}$ only differs from S^{W_i} by $\leq L$ edges. Our goal is to calculate the probability of the event $\mathcal{E} = \cup_Y \mathcal{E}_Y$, which is the union on all combinations of k nodes.

We first compute the probability of individual $\mathcal{E}_Y$. Recall that the edges between $\binom{k}{2} - (k-1)$ node pairs in S^{W_i} are generated randomly with probability $\frac{1}{2}$ and are independent of G^{W_j}, while the rest $k-1$ edges ($< x_l, x_{l+1} >, l = 1...k-1$) are fixed. Thus we can show that the probability that a subgraph $G^{W_j}[Y]$ differs from S^{W_i} by h edges is upper bounded by $\frac{1}{2}^{e-k+1} \cdot \binom{e}{h}$ where $e = \binom{k}{2}$. Therefore, we can derive the probability of $\mathcal{E}_Y$ as $P(\mathcal{E}_Y) \leq \frac{1}{2}^{e-k+1} \cdot \sum_{h=0}^{L} \binom{e}{h}$. And consequently, we have $P(\mathcal{E}) \leq n^k \cdot \frac{1}{2}^{e-k+1} \cdot \sum_{h=0}^{L} \binom{e}{h}$, where $e = \binom{k}{2}$, $k = (2+\delta) log_2 n$, and n is the node count of G^{W_j}.

Next, given the probability of uniqueness $1 - P(\mathcal{E})$, we compute the upper bound on L to ensure $1 - P(\mathcal{E}) \geq 0.99999$ for all the graphs in Table 1 except Road graphs, Co-purchasing graphs, and P2P network graphs. Again we set $\delta = 0.3$. The result is listed in Table 3, where the maximum limit of L varies between 0 and 12. In general, the larger the graph, the higher the upper bound on L.

7. EXPERIMENTAL EVALUATION

We use real network graphs to evaluate the performance of the graph watermarking system in three key metrics: *false positives*, *graph distortion* and *watermark robustness*. Having analytically quantified watermark uniqueness in §5 and §6, we focus on examining graph distortion and watermark robustness while ensuring $\leq 0.001\%$ false positive rate. We also study the computational efficiency of the proposed watermark embedding and extraction schemes.

Experiment Setup. Given the large number of graph computations per data point, we focus our experiments on two larger network graphs in Table 1, *i.e.* the LA regional Facebook graph and the Flickr graph. The two graphs have very different sizes and graph structures. To guarantee $\leq 0.001\%$ false positives, we use $\delta = 0.3$, $k = 45$ for the LA graph, and $\delta = 0.3$, $k = 48$ for the

Graph	Oregon (1)	Oregon (2)	CAIDA	Email (Enron)	arXiv (Theo. Cit.)
L Bound	0	1	1	1	1
Graph	arXiv (Phy. Cit.)	arXiv (Phy.)	arXiv (Astro)	Patent	Slashdot (08/11/06)
L Bound	1	1	1	2	3
Graph	Twitter	Slashdot (09/02/16)	Slashdot (09/02/21)	Slashdot (09/02/22)	Brightkite
L Bound	3	3	3	3	3
Graph	Russia	Epinions (1)	Google+	Epinions (2)	Standford
L Bound	4	4	4	5	5
Graph	Email (Europe)	Gowalla	BerkStand	DBLP	NorteDame
L Bound	5	5	6	7	7
Graph	L.A.	London	Flickr	Wiki	Google
L Bound	8	8	8	8	8
Graph	Skitter	Youtube	Pokec	arXiv (Condense)	Livejournal
L Bound	8	9	9	11	12

Table 3: Upper bound of L for the 35 network graphs.

Graph	Nodes (%)	Edges (%)	dK-2 Deviation
Watermarked LA	0.037%	0.033%	0.0008
Watermarked Flickr	0.014%	0.019%	0.0001

Table 4: Percentage of modified nodes/edges after embedding 5 watermarks into a graph and dK-2 Deviation.

Flickr graph. For our basic design, we generate 1 watermark per graph. For our advanced design, we set L to 8, the degree bucket size to 10, and the NSD similarity threshold to $\theta = 0.75$. For each user, we embed 5 watermarks in its graph, 3 individual watermarks and 2 group watermarks. We chose these settings because they work well in practice. We leave the optimization of these parameters to future work.

Next, we present experimental results on graph distortion, robustness against attacks, and computational efficiency.

7.1 Graph Distortion from Watermarks

We consider three metrics to measure graph distortion.

- *Modifications to the raw graph* – We count the number of nodes/ edges modified by embedding watermarks.
- *dK-2 Deviation* – dK-2 series, *i.e.* joint degree distributions, are an important graph structural metric [35]. We quantify graph distortion using the normalized Euclidean distance between the dK-2 series of the original and of the watermarked graphs [4].
- *Graph metrics w/ and w/o watermarks* – We measure the widely used graph metrics before and after the watermarking, including degree distribution, assortativity (AS) [35], clustering coefficient (CC) [35], average path length, and diameter. Large deviation in any of the metrics indicates large distortion.

We have examined the distortion introduced by both the basic and advanced designs. We only show the results of the advanced design because it adds more watermarks and thus leads to higher distortion. For LA and Flickr graphs, we generate 10 different watermarked graphs (using 10 different seeds) and present the average result across these graphs. Because computing shortest paths on the large graphs is highly computational intensive, we compute the average path length and diameter among 1000 random nodes [40].

Table 4 shows the percentage of modified nodes/edges by watermarking. Even after embedding 5 watermarks, the modification for both graphs is less than 0.04%, implying little distortion on the watermarked graphs. This is further confirmed by the average dK-2 distances. We also compare the original and watermarked graphs using 5 graph metrics: AS, CC, degree distribution, average path length, and diameter. Similarly, the metrics remain the same before and after watermarking, and we found no difference between the statistical distributions of each metric in the graphs.

Together, this indicates that embedding watermarks produces negligible impact on graph structure. Thus we believe watermarked graphs can replace the originals in graph applications and produce (near-)identical results.

[4]The Euclidean distance between dK-2 series is normalized by the number of tuples in the dK-2 series.

7.2 Robustness against Attacks

Next, we study the robustness of the watermarking system under the attacks. For each of the two attacks discussed in §6.1, we vary the attack strength, repeat each experiment 10 times, and examine the following two metrics:

- *Robustness* – In the single attacker model, the robustness is the ratio of graphs from which we can successfully extract at least one of the 3 individual watermarks. In the collusion attack, in addition to this ratio, we also measure the ratio of graphs where we can extract at least one of the 5 watermarks (3 individual + 2 group watermarks).
- *Cost of the attack* – The normalized distortion on the attacked graphs. It represents the dK-2 deviation between the attacked graphs and the original graph, normalized by that between the "clean" watermarked graphs and the original graph. If the normalized distortion is > 1, the attack introduces more distortion than embedding watermarks.

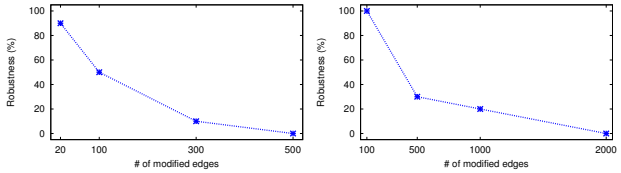
Results on the Single Attacker Model. For the single attacker model, we quantify the attack strength by the number of modified edges. The robustness and the cost of the attack are measured as a function of the number of modified edges.

We first evaluate the robustness of the basic watermark. Figure 2 shows that randomly modifying a small number of edges disrupted the extraction process. For example, in LA, our basic design cannot recover the watermark with 100% probability when we only modify 20 edges. In each case, at least one of the nodes in the watermarks had a modified NSD label (one of its neighbors' degree changed), and it could not be located in the extraction process. We show the distortion on the attacked graphs in Figure 3. As expected, the small number of modifications causes small distortions in graph structures. Still in LA, when the robustness is 0, the distortion is around 3x more than that the watermarked graphs. Both results show that the basic watermark scheme is easily disrupted by small, single user attacks.

Figure 4 shows that robustness of the improved scheme decreases with attack strength, since more edges are modified to "destroy" watermarks. Like in Flickr, the system can handle attack strength up to 933K modified edges, which is $>$ 400x stronger than the maximum attack strength in the basic design. On the other hand, Figure 5 shows that the cost of these attacks is large. For Flickr, with more than 1.4M modified edges, an attack leads to 800x more distortions over that caused by embedding 5 watermarks. Our improved watermark is highly robust against single user attacks.

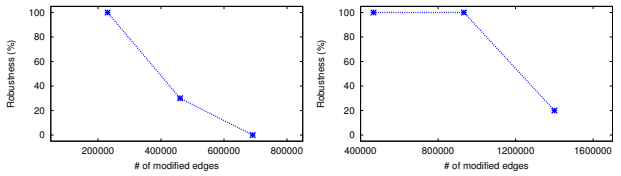
Results on Collusion Attacks. To implement the collusion attack described in §6.1, we first generate 10 watermarked graphs and randomly pick M_a graphs from them as the graphs acquired by the attacker. We vary the number of graphs obtained by the attacker M_a between 2 to 5. For each M_a value we repeat the experiments 10 times and report the average value. Since basic watermarks are easily disrupted by the collusion attack, we focus on the robustness of the improved mechanisms.

Figure 6(a)-(b) shows the robustness of the watermarked LA and Flickr graphs against the collusion attack. Figure 6(a) shows that in



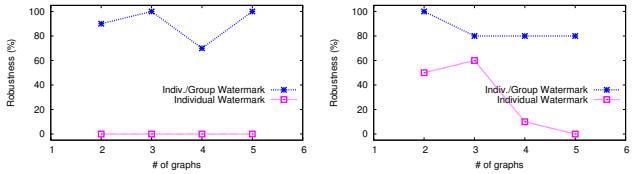
(a) Robustness, LA basic (b) Robustness, Flickr basic

Figure 2: The robustness of the basic design against the single attacker model.



(a) Robustness, LA improve (b) Robustness, Flickr improve

Figure 4: The robustness in the improved design against the single attacker model.



(a) Robustness, LA (b) Robustness, Flickr

Figure 6: The robustness against the collusion attack.

Graph	Embedding (s)	Basic Extraction		Improved Extraction	
		Single(s)	Parallel (s)	Single (s)	Parallel (s)
LA	40	270	39	310	42
Flickr	80	767	195	776	197
Livejournal	695	2568	310	2605	317

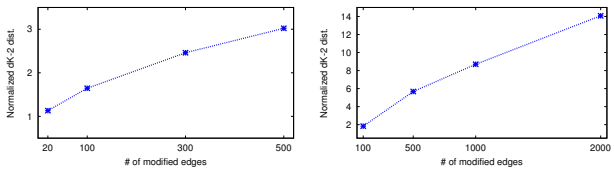
Table 5: The efficiency of the watermarking system.

LA, by applying majority votes on raw edges, the collusion attack can effectively remove all 3 individual watermarks. However, the attack is ineffective in removing both group watermarks: we can extract at least one group watermark in more than 60% of the attacked graphs. Here the robustness values deviate slightly from that projected by Equation (2) because we limit the number of statistical sampling to 10 runs. Unlike LA, Figure 6(b) plots that the collusion attack cannot remove all the individual watermarks in Flickr when using 2 or 3 watermarked graphs. This is because the deanonymization method causes a large portion of nodes mismatched in Flickr (30% nodes). Finally, Figure 7 shows that the collusion attacks also introduce larger distortions in graph structure.

These results show that even a powerful collusion attack is ineffective in removing all embedded watermarks. Moreover, the potential inaccuracy of the deanonymization method makes the attack even weaker in removing individual watermarks. Of course, the attackers will eventually succeed in disrupting watermarks if they are willing to modify and sacrifice the utility of the graph. While we provide a robust defense against attackers with low level of tolerance for graph distortion, we hope follow-on work will develop more robust defenses against higher distortion attacks.

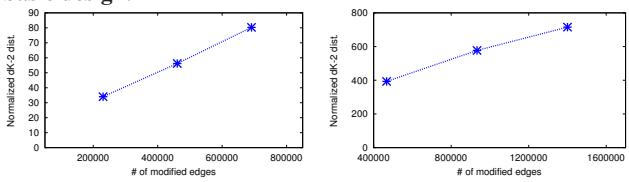
7.3 Computational Efficiency

We measure the efficiency of embedding and extracting watermarks, including the time to select candidates (step 2) and to identify watermarks (step 3). We accelerate the extraction process by



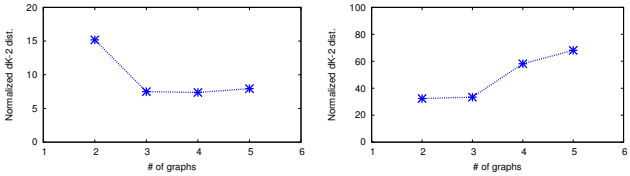
(a) Distortion, LA basic (b) Distortion, Flickr basic

Figure 3: The distortion caused by the single attacker model in the basic design.



(a) Distortion, LA improve (b) Distortion, Flickr improve

Figure 5: The distortion caused by the single attacker model in the improved design.



(a) Distortion, LA (b) Distortion, Flickr

Figure 7: The distortion caused by the collusion attack.

parallelizing the key steps across servers. When a watermark is found or no more candidates are unchecked, the extraction process stops (for that user).

We perform measurements to quantify impact of parallelizing extraction over a cluster. All system parameters are the same as previous tests, except that we embed 1 watermark into a graph. We compare the improved watermark extraction method to the basic extraction method. In addition to Flickr and LA graphs, we also measure the efficiency on Livejournal graph [26], a larger graph with 5.2M nodes, 49M edges. We parallelize watermark extraction across 10 servers, each with 192GB RAM, and report the average times from 10 different watermarked graphs.

Table 5 shows that our system is efficient in embedding and extracting watermarks. Embedding one watermark into a graph is very fast, *e.g.* average embedding time for the largest graph, Livejournal, is around 12 minutes. Even using one server to extract watermarks, the computation time is small, *e.g.* 13 minutes in Flickr using both the basic and improved schemes. Time to identify the watermark graph on the candidate subgraphs (step 3) is much less than the time required to find candidates (step 2), which corresponds to 99% of total computation time. Since finding candidates takes $O(kn)$ computational complexity and $k = (2+\delta)\log_2 n$, the complexity to extract a watermark from a real-world graph is $O(n \log_2 n)$. Here k is node number in the watermark graph and n is nodes in the total graph.

Second, we find that distributed extraction produces good speedup, 8 over 10 servers for Livejournal and 7 for LA (for both extraction methods). The speedup for Flickr is only around 4 using both methods, because one of the watermarked graphs takes much longer time than others in finding candidates, 4x longer. Without this outlier, the average parallel extraction time on Flickr is around 2.5 minutes for both methods, 5x faster than using single server.

Finally, there is no significant difference between computation time for the two extraction methods.

8. CONCLUSION

In this paper, we take a first step towards the design and implementation of a robust graph watermarking system. Graph watermarks have the potential to significantly impact the way graphs are shared and tracked. Our work identifies the critical requirements of such a system, and provides an initial design that targets the critical properties of uniqueness, robustness to attacks, and minimal distortion to the graph structure. We also identify key attacks against graph watermarks, and evaluate them against an improved design with additional features for improved robustness under attack. Finally, we show the watermarking system is efficient in both watermark embedding and watermark extraction.

Acknowledgments

The authors wish to thank the anonymous reviewers and our shepherd Daniel Figueiredo for their helpful comments. This work is supported in part by NSF grants IIS-1321083, CNS-1224100, CNS-1317153 and CNS-1527939.

9. REFERENCES

[1] AGRAWAL, R., AND KIERNAN, J. Watermarking relational databases. In *Proc. of VLDB* (2002).

[2] ALBERT, R., JEONG, H., AND BARABÁSI, A.-L. Internet: Diameter of the world-wide web. *Nature 401*, 6749 (1999), 130–131.

[3] ARRINGTON, M. How "dirty" mp3 files are a back door into cloud drm. TechCrunch, April 2010.

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

[5] BENDER, W., GRUHL, D., MORIMOTO, N., AND LU, A. Techniques for data hiding. *IBM systems journal* (1996), 313–336.

[6] CHO, E., MYERS, S. A., AND LESKOVEC, J. Friendship and mobility: user movement in location-based social networks. In *KDD* (2011).

[7] COLLBERG, C. S., KOBOUROV, S. G., CARTER, E., AND THOMBORSON, C. D. Graph-based approaches to software watermarking. In *WG* (2003).

[8] COOK, S. A. The complexity of theorem-proving procedures. In *STOC* (1971).

[9] GILBERT, E. N. Random graphs. *The Annals of Mathematical Statistics* (1959), 1141–1144.

[10] HANHIJÄRVI, S., GARRIGA, G. C., AND PUOLAMÄKI, K. Randomization techniques for graphs. In *SDM* (2009).

[11] HAY, M., MIKLAU, G., JENSEN, D., TOWSLEY, D., AND WEIS, P. Resisting structural re-identification in anonymized social networks.

[12] HAY, M., MIKLAU, G., JENSEN, D., WEIS, P., AND SRIVASTAVA, S. Anonymizing social networks. Tech. Rep. 07-19, UMass, 2007.

[13] KAMRAN, M., ET AL. A robust, distortion minimizing technique for watermarking relational databases using once-for-all usability constraints. *IEEE TKDE* (2012).

[14] LEE, S.-J., AND JUNG, S.-H. A survey of watermarking techniques applied to multimedia. In *ISIE* (2001).

[15] LESKOVEC, J., ADAMIC, L. A., AND HUBERMAN, B. A. The dynamics of viral marketing. *TWEB* (2007).

[16] LESKOVEC, J., ET AL. Statistical properties of community structure in large social and information networks. In *WWW* (2008).

[17] LESKOVEC, J., HUTTENLOCHER, D., AND KLEINBERG, J. Predicting positive and negative links in online social networks. In *WWW* (2010).

[18] LESKOVEC, J., KLEINBERG, J., AND FALOUTSOS, C. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD* (2005).

[19] LESKOVEC, J., KLEINBERG, J., AND FALOUTSOS, C. Graph evolution: Densification and shrinking diameters. *TKDD* (2007).

[20] LESKOVEC, J., LANG, K. J., DASGUPTA, A., AND MAHONEY, M. W. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics 6*, 1 (2009), 29–123.

[21] LESKOVEC, J., AND MCAULEY, J. J. Learning to discover social circles in ego networks. In *Advances in neural information processing systems* (2012), pp. 539–547.

[22] LI, Y., SWARUP, V., AND JAJODIA. Fingerprinting relational databases: Schemes and specialties. *IEEE TDSC 2*, 1 (2005), 34–45.

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

[24] MACQ, B. M., AND QUISQUATER, J.-J. Cryptology for digital tv broadcasting. *Proceedings of the IEEE* (1995), 944–957.

[25] MENEZES, A. J., OORSCHOT, P. V., AND VANSTONE, S. *Handbook of Applied Cryptography*. CRC Press, Inc., 1996.

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

[27] NARAYANAN, A., AND SHMATIKOV, V. De-anonymizing social networks. In *Proc. of IEEE S&P* (May 2009).

[28] NARAYANAN, A., AND SHMATIKOV, V. Link prediction by de-anonymization: How we won the kaggle social network challenge. In *IJCNN* (2011).

[29] OHBUCHI, R., UEDA, H., AND SHUH, E. Robust watermarking of vector digital maps. In *ICME* (2002).

[30] OHBUCHI, R., UEDA, H., AND SHUH, E. Watermarking 2d vector maps in the mesh-spectral domain. In *Shape Modeling International* (2003).

[31] QU, G., AND POTKONJAK, M. Analysis of watermarking techniques for graph coloring problem. In *ICCAD* (1998).

[32] RICHARDSON, M., AGRAWAL, R., AND DOMINGOS, P. Trust management for the semantic web. In *The Semantic Web-ISWC 2003*. Springer, 2003, pp. 351–368.

[33] ROBSHAW, M. J. B. MD2, MD4, MD5, SHA and other hash functions. Tech. Rep. TR-101, RSA Laboratories, 1995. v. 4.0.

[34] RUANAIDH, J. O., DOWLING, W., AND BOLAND, F. Phase watermarking of digital images. In *ICIP* (1996).

[35] SALA, A., CAO, L., WILSON, C., ZABLIT, R., ZHENG, H., AND ZHAO, B. Y. Measurement-calibrated graph models for social network experiments. In *Proc. of WWW* (2010).

[36] SALA, A., ZHAO, X., WILSON, C., ZHENG, H., AND ZHAO, B. Y. Sharing graphs using differentially private graph models. In *IMC* (2011).

[37] STEVE, W. Information authentication for a slippery new age. *Dr. Dobbs Journal* (1995), 18–26.

[38] TAKAC, L., AND ZABOVSKY, M. Data analysis in public social networks. In *International Scientific Conference AND International Workshop Present Day Trends of Innovations* (2012).

[39] VENKATESAN, R., VAZIRANI, V., AND SINHA, S. A graph theoretic approach to software watermarking. In *Information Hiding* (2001).

[40] WILSON, C., SALA, A., PUTTASWAMY, K. P. N., AND ZHAO, B. Y. Beyond social graphs: User interactions in online social networks and their implications. *ACM Trans. on the Web 6*, 4 (2012).

[41] WOLFE, G., WONG, J. L., AND POTKONJAK, M. Watermarking graph partitioning solutions. In *DAC* (2001).

[42] XIA, X.-G., BONCELET, C. G., AND ARCE, G. R. A multiresolution watermark for digital images. In *ICIP* (1997).

[43] YANG, J., AND LESKOVEC, J. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics* (2012), ACM, p. 3.

[44] YING, X., AND WU, X. Randomizing social networks: a spectrum preserving approach. In *SDM* (2008).

[45] ZHAO, X., LIU, Q., ZHOU, L., ZHENG, H., AND ZHAO, B. Y. Graph watermarks. *Arxiv preprint arXiv:1506.00022* (2015).

[46] ZHOU, B., ET AL. Preserving privacy in social networks against neighborhood attacks. In *Proc. of ICDE* (2008).

[47] ZHU, W., THOMBORSON, C., AND WANG, F.-Y. A survey of software watermarking. In *ISI*. 2005.

[48] ZOU, L., CHEN, L., AND ÖZSU, M. T. K-automorphism: A general framework for privacy preserving network publication. In *VLDB* (2009).