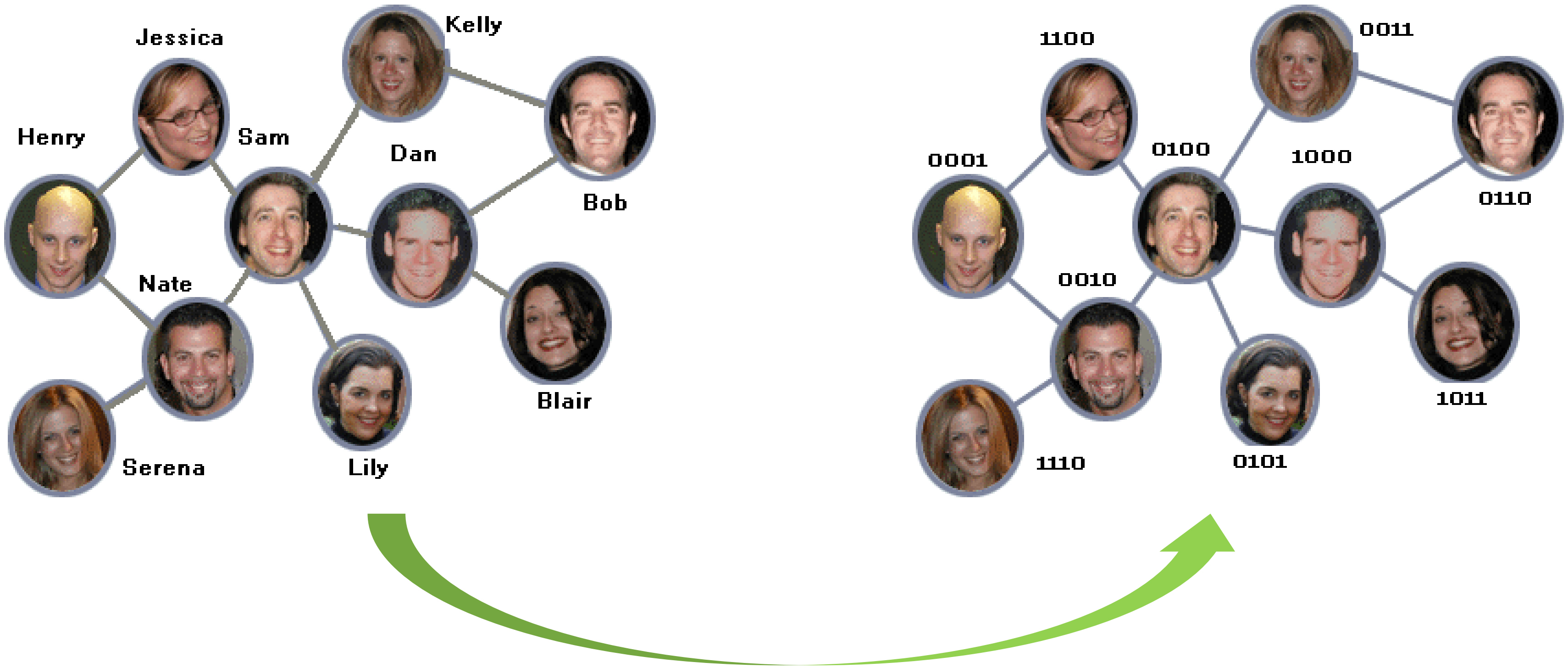
Wherefore Art Thou R3579X? Anonymized Social Networks, Hidden Patterns and Structural Steganegraphy

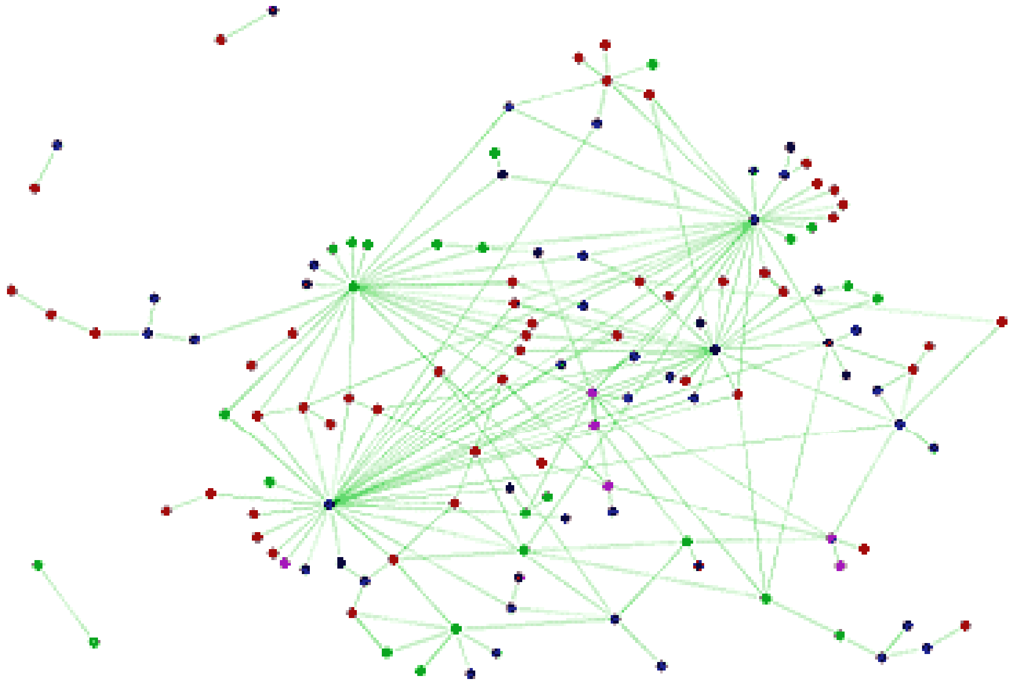
Lars Backstrom, Cynthia Dwork, Jon Kleinberg

Anonymized Networks



1. For researchers, names are meaningless
2. Users also need privacy

Anonymized Network(cont')



Release just a big random labeled graph
with millions of edges!
No other conceptual data released!

Attack Scenario

1. Attacker may add a few new nodes prior to release.

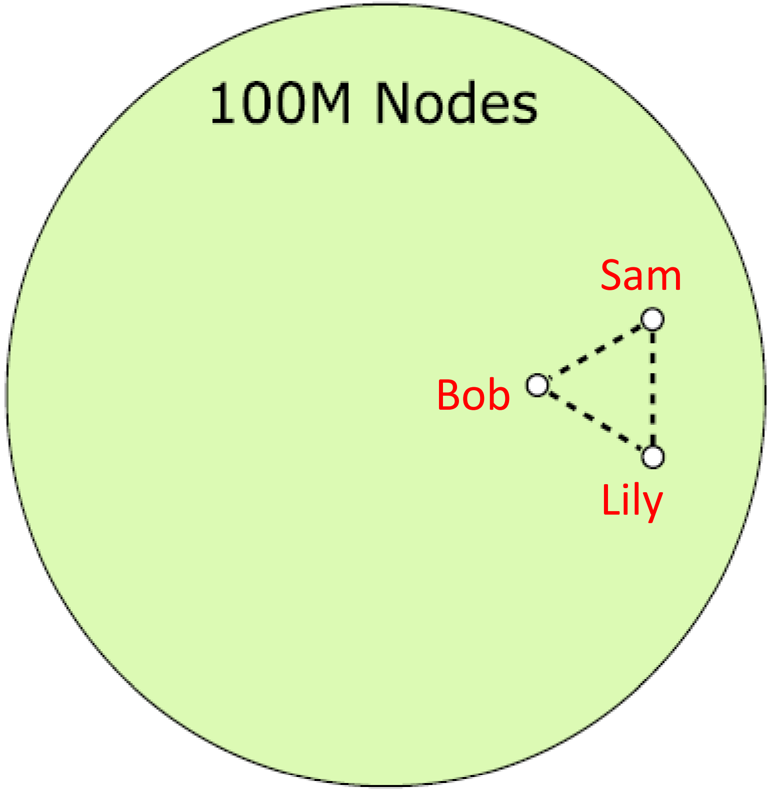
   - Would like to add as few as possible

2. Attacker may add edges from these few nodes to any nodes in the network

   - May link to any person whose 'name' he knows

3. Goal of the attack is to discover if two named users are connected, i.e. their connections

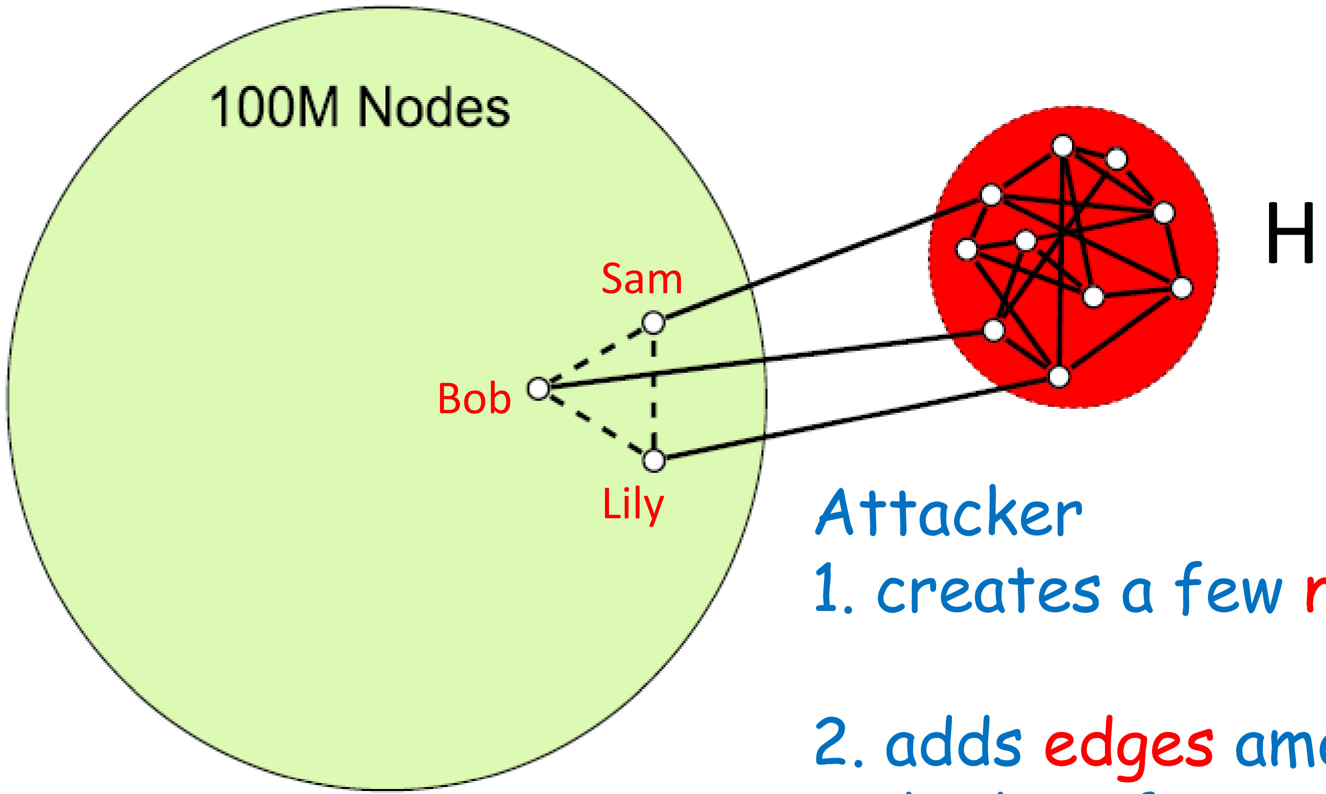
Attack Structure



Before release of the network,

the attacker finds some targeted users (named users)

Attack Structure(cont')

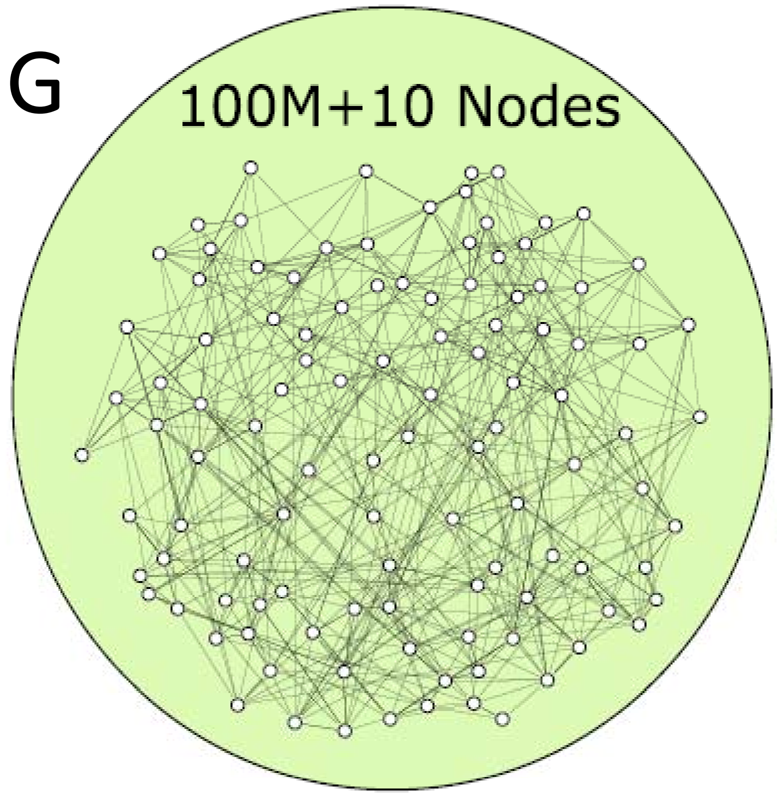


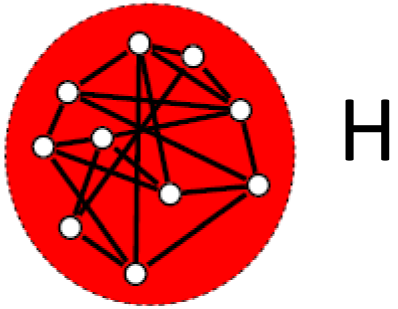
Attacker

1. creates a few new nodes

2. adds edges among them and edges from the new nodes to the named nodes

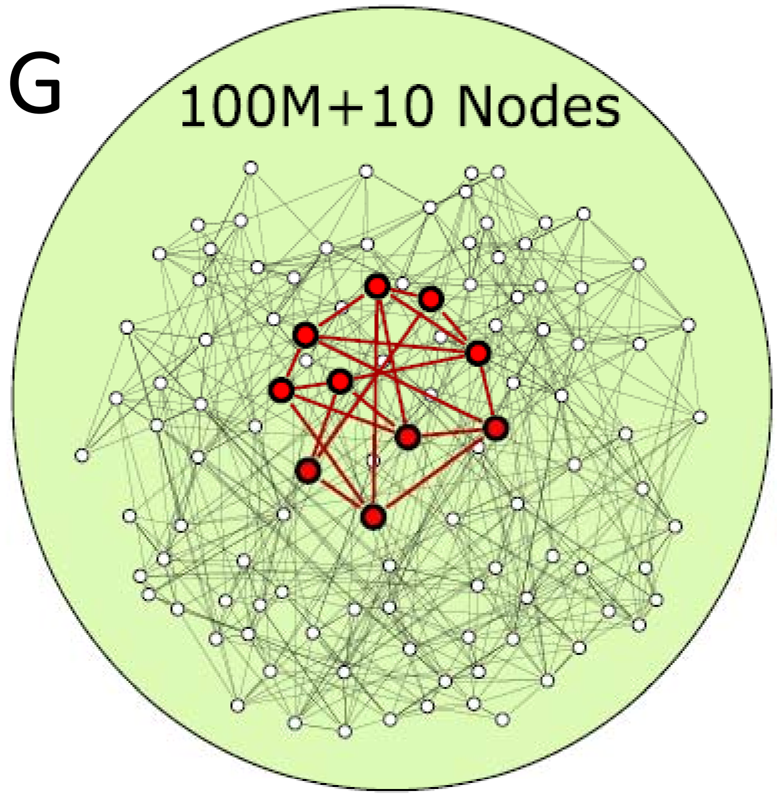
Attack Structure(cont')

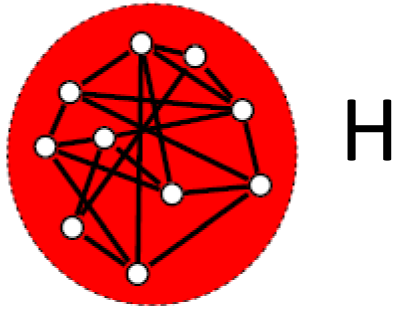




Network is released to public!

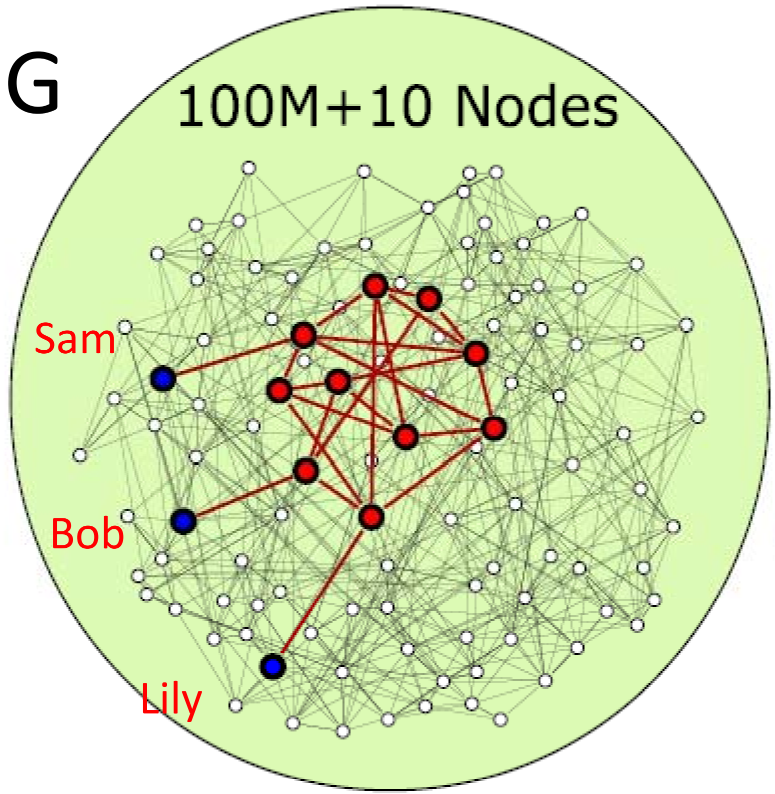
Attack Structure(cont')





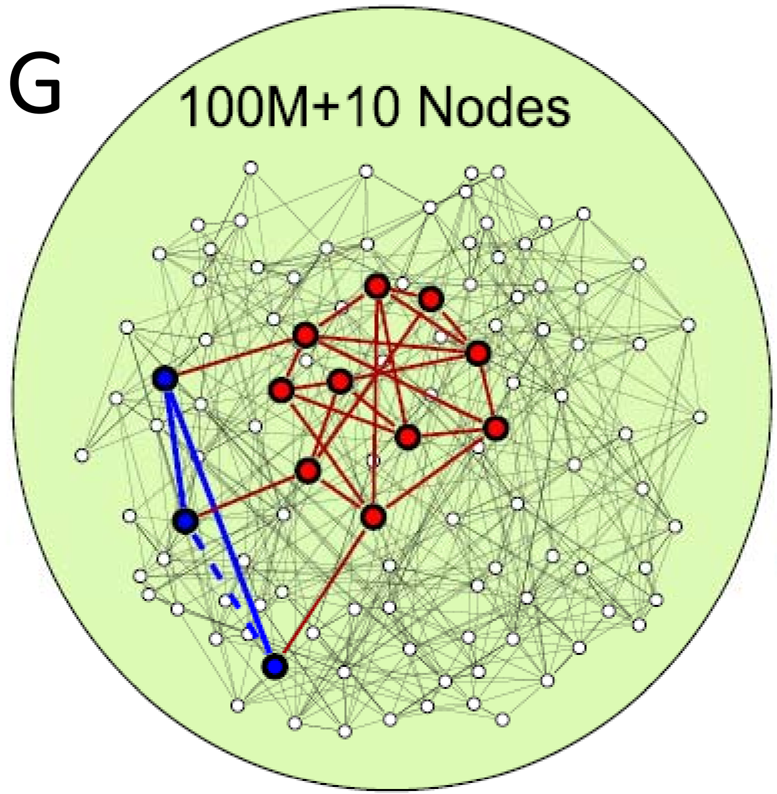
Attacker
locates
the inserted H

Attack Structure(cont')



Follow links to identify named users

Attack Structure(cont')



Follow links to identify named users

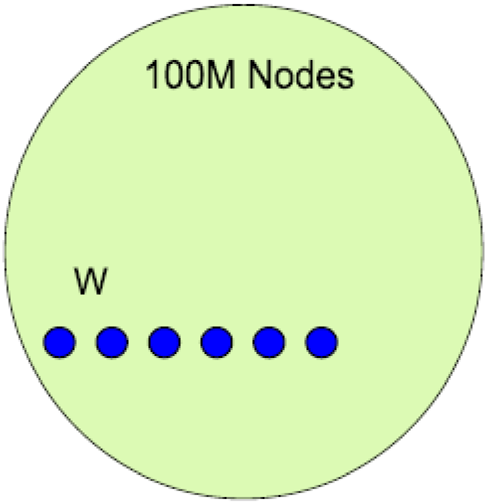
Attacker then determines which pairs are connected!
Done!

How can the attacker

locates

the inserted H

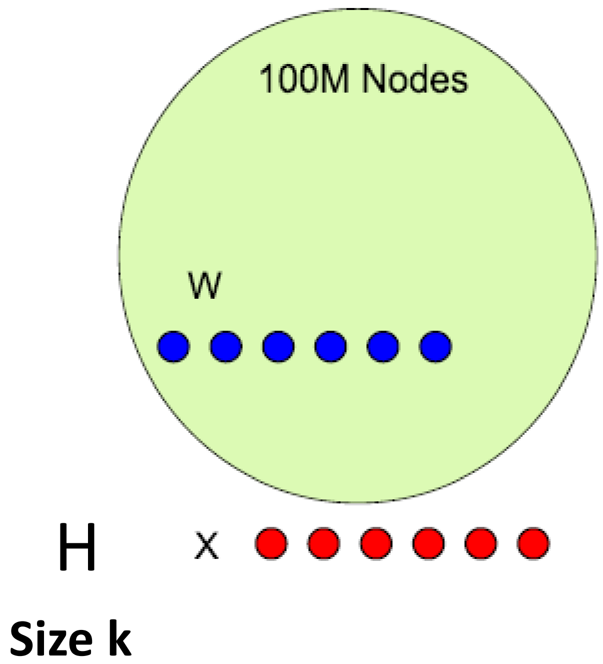
Walk-Based Attack



1. Attacker chooses k named users

   W = {w1,w2, . . . ,wk}

Walk-Based Attack



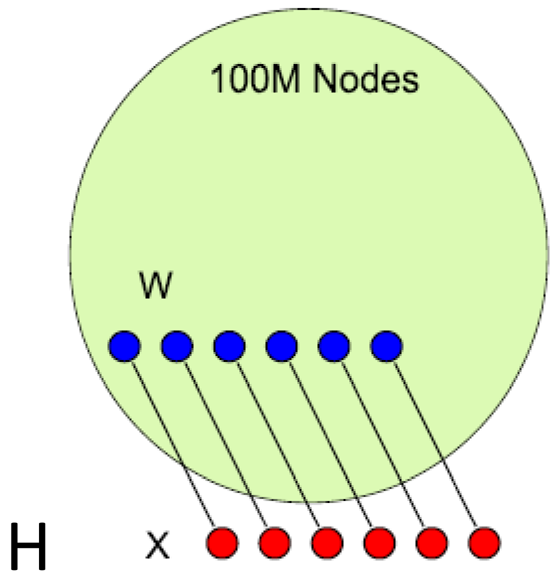
1. Attacker chooses k named users

   W = {w1,w2, . . . ,wk}

2. Creates k new nodes

   X = {x1, x2, . . . , xk}

Walk-Based Attack



1. Attacker chooses k named users

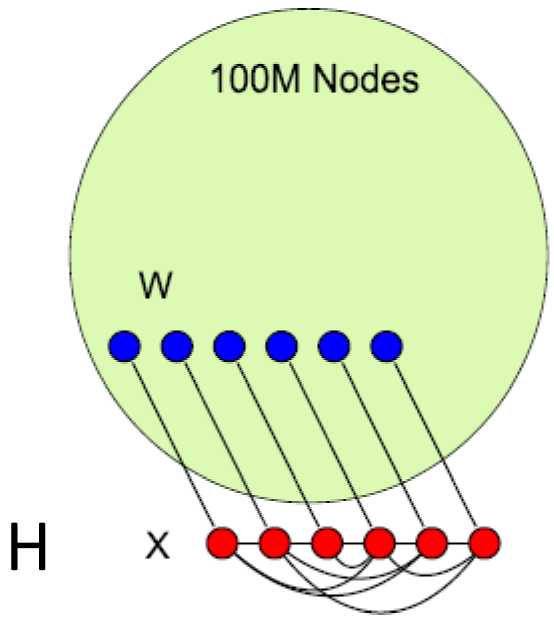
   W = {w1,w2, . . . ,wk}

2. Creates k new nodes

   X = {x1, x2, . . . , xk}

3. Creates edges (xi,wi)

Walk-Based Attack



1. Attacker chooses k named users

   W = {w1,w2, . . . ,wk}

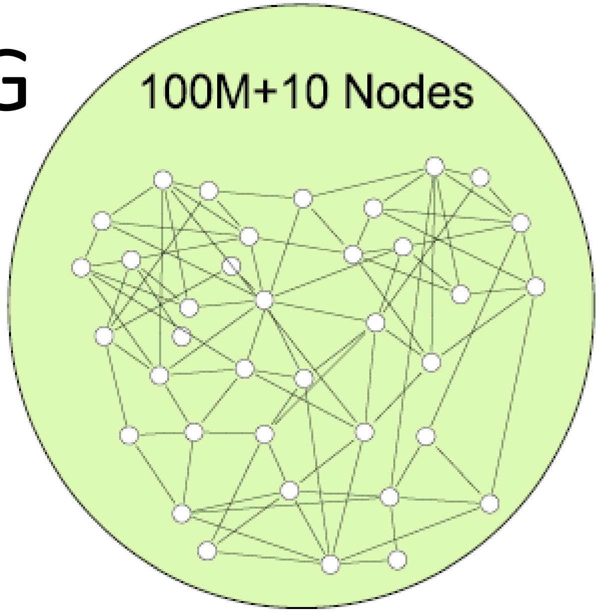
2. Creates k new nodes

   X = {x1, x2, . . . , xk}

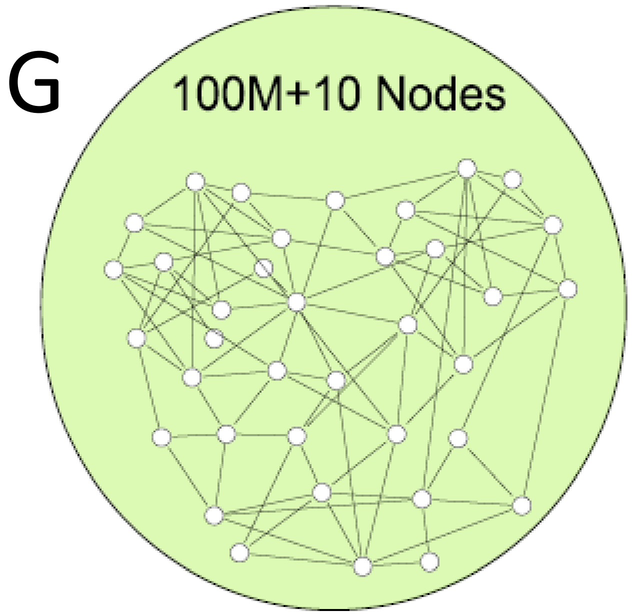
3. Creates edges (xi,wi)

4. Creates each edge (xi, xi+1) – Hamiltonian Path and each other edge with probability 0.5

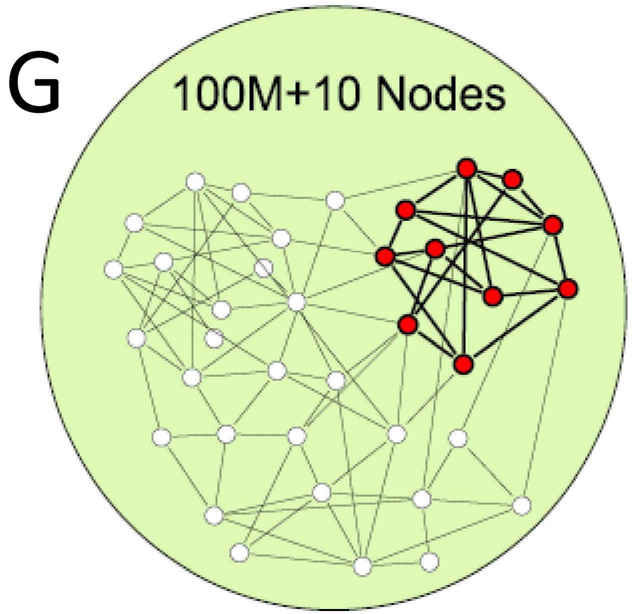
G



Is the attacker able to **locate** the inserted H now

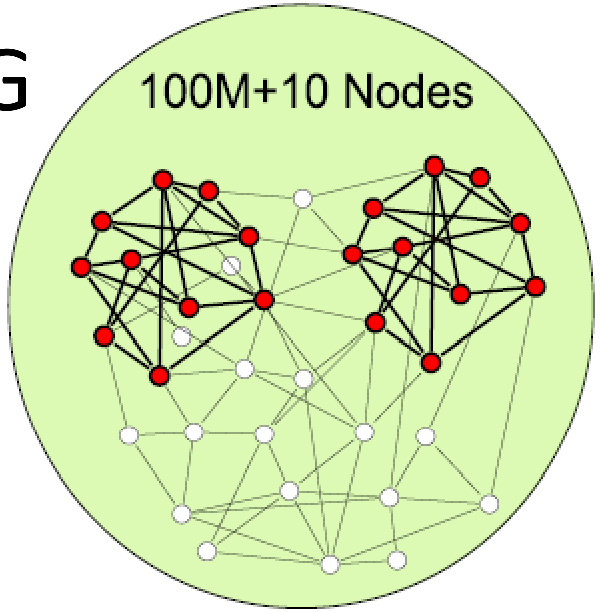


This might not work!



Why this might not work?

G



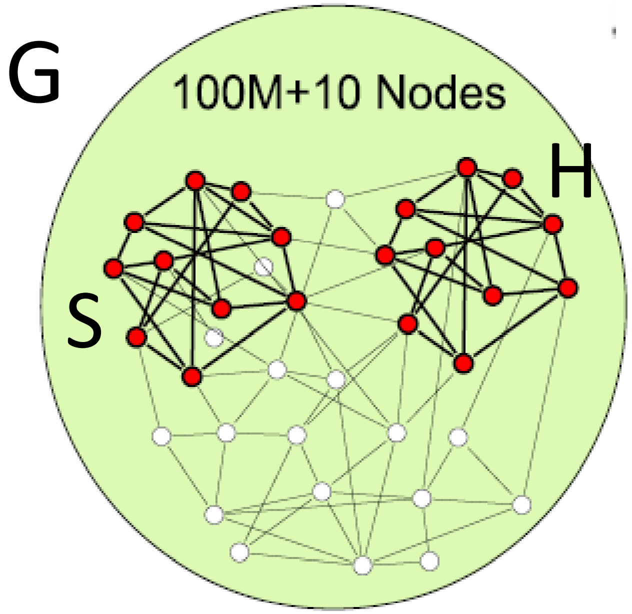
H?

Why this might not work?

More than one matches!

No way to differentiate!

Success Conditions



Uniqueness

- No S != X such that G[S] and G[X] = H are isomorphic
- H has no non-trivial automorphisms

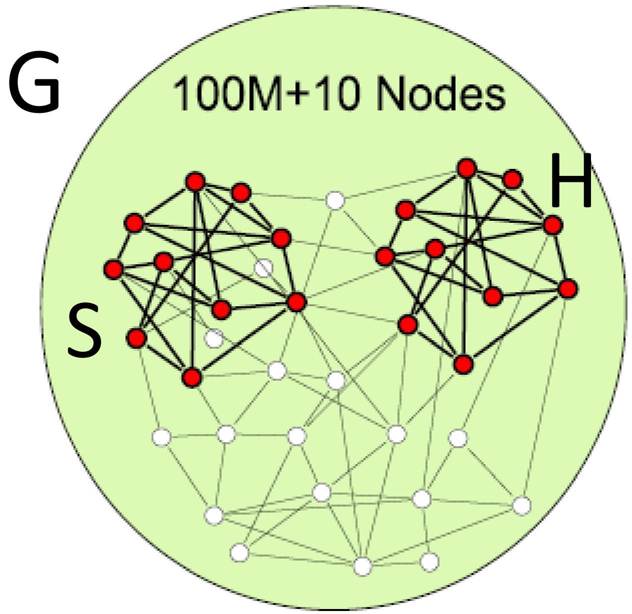
Intuitively,

Isomorphism: without labeling, two graphs are actually the same

Automorphism: a graph has internal symmetry

(relabeling its nodes preserves graph's structure)

Success Conditions(cont')



Uniqueness

- No S != X such that G[S] and G[X] = H are isomorphic
- H has no non-trivial automorphisms

Recoverability

- Find H efficiently, given G

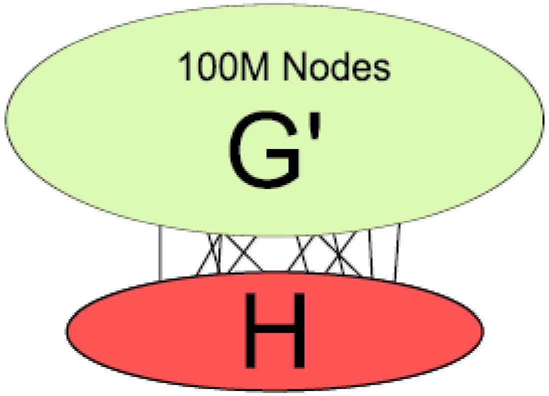
Uniqueness Proof

Intuition:

- Erdös showed that a random graph will not contain certain non-random subgraphs

- We need to show that a non-random graph will not have a certain random subgraph

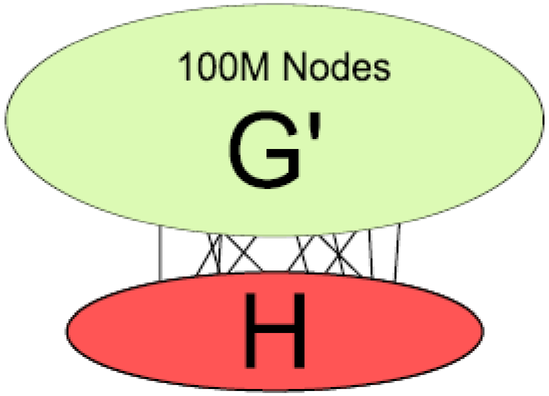
Uniqueness Proof



Proof technique:

1. Count the number of size k subgraphs in $G' = G - H$

Uniqueness Proof

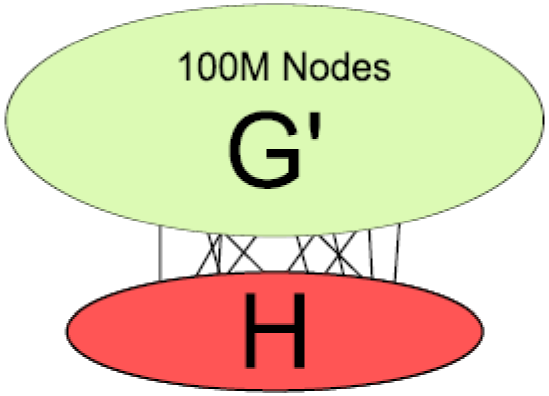


Proof technique:

1. Count the number of size k subgraphs in G′ = G – H

2. Find probability that each one is a match

Uniqueness Proof



Proof technique:

1. Count the number of size k subgraphs in $G' = G - H$

2. Find probability that each one is a match

3. Take union-bound to show that there is no match with high probability

$$P\left(\bigcup_i A_i\right) \leq \sum_i P(A_i).$$

Uniqueness Proof(cont')

1. In G', number of labeled subgraph of size k,

$$\binom{n-k}{k} k! \;\le\; n^k$$

2. Probability that a particular labeled subgraph matches H is no more than

$$\Pr[\mathcal{E}_S] \;=\; 2^{-\binom{k}{2}+(k-1)} = 2^{-\binom{k-1}{2}}$$

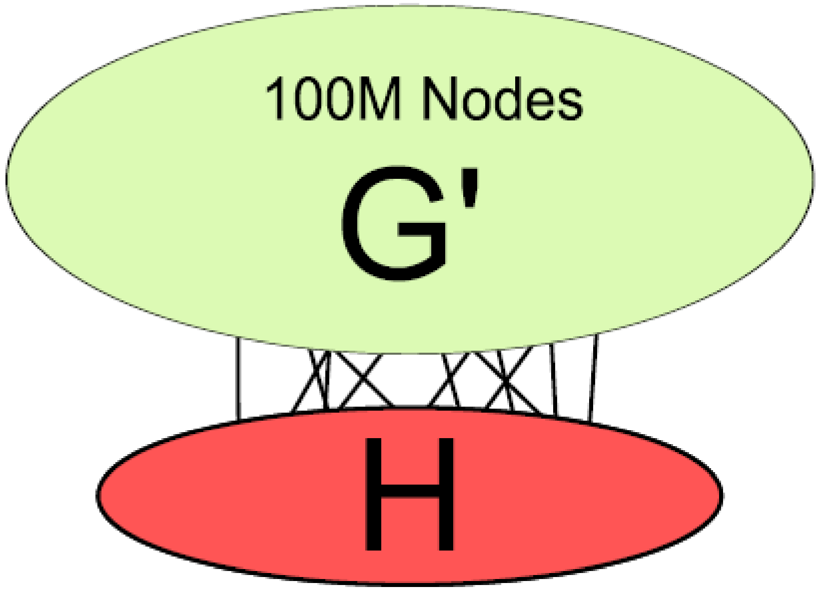
3. Probability that there is at least a match in these subgraphs is (use union bound) $\mathcal{E} \;=\; \cup_S \mathcal{E}_S$

$$\Pr[\mathcal{E}] \;<\; n^k \cdot 2^{-\binom{k-1}{2}}$$

$$\le\; 2^{[-\delta k^2/2(2+\delta)]+3k/2+1}$$

It goes to 0 exponentially quickly in k !

Uniqueness Proof(cont')



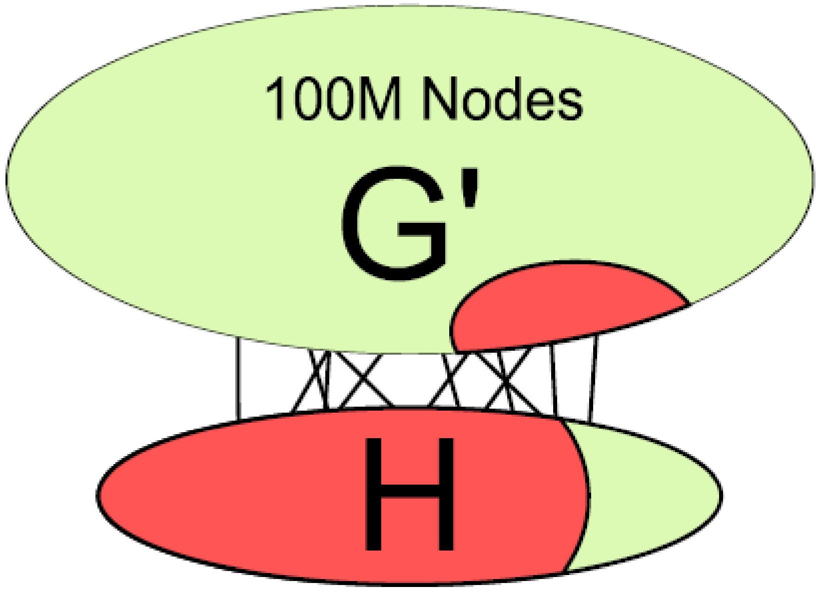
Intuition:

1. k is small, high probability that there's match in G';
2. k is large, low probability.

But we want k to be relatively small, so we have to find a Balance here!

We choose $k \doteq (2+\delta)\log n$ for a small constant $\delta > 0$

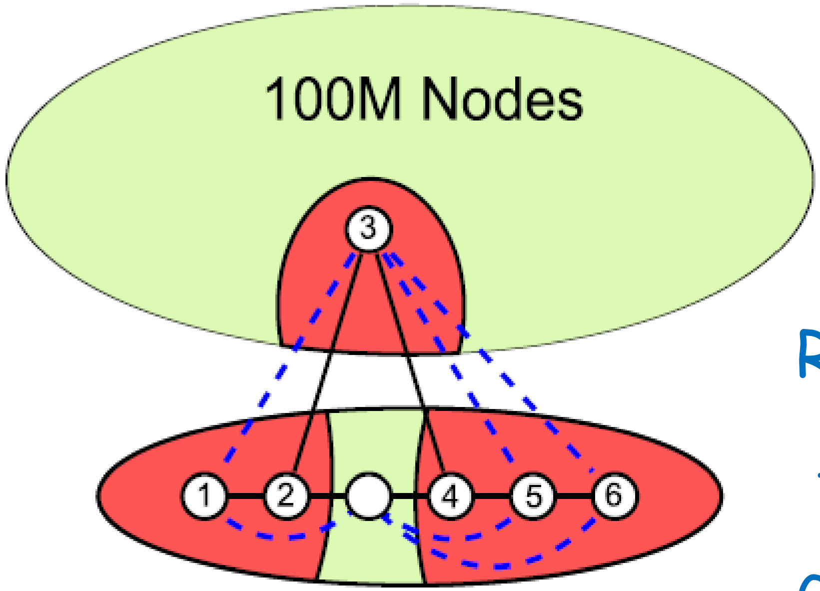
Uniqueness Proof(cont')



Showed that *G'* has
no copies of H
(with high probability).

However,
attaching H to *G'* may make a copy in *G*.

Uniqueness Proof(cont')



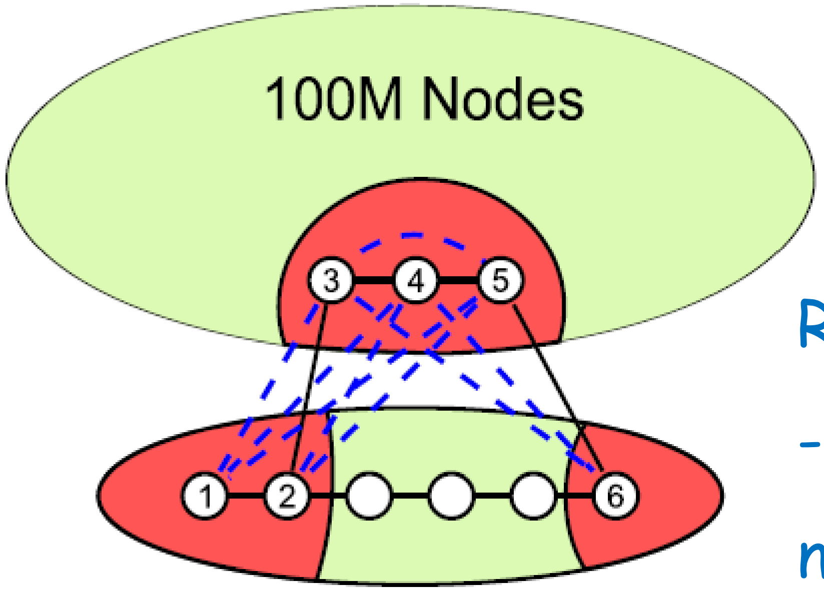
Replace a few nodes from H

-> nodes from G' must have

correct connections to H

-> Unlikely

Uniqueness Proof(cont')



Replace many nodes from H

-> internal connections of

nodes must be correct as well

-> More Unlikely

Analysis uses similar techniques, but is more complex!

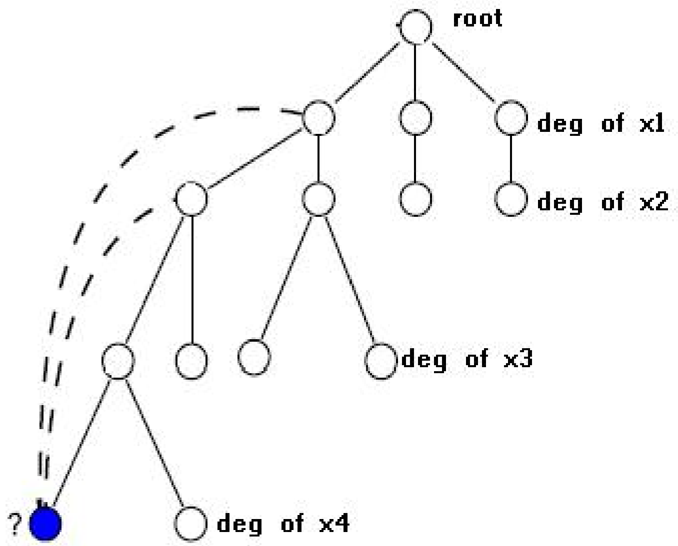
How to find the unique H in G?

How to find the unique H in G?

Recovery Algorithm

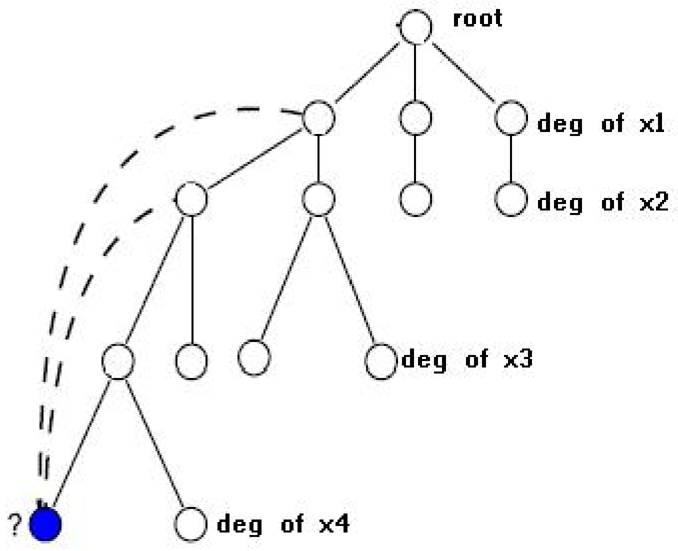
Brute force with pruning

Recovery Algorithm



1. Start from root, pick all the nodes that have the degree of x1, to be root's children

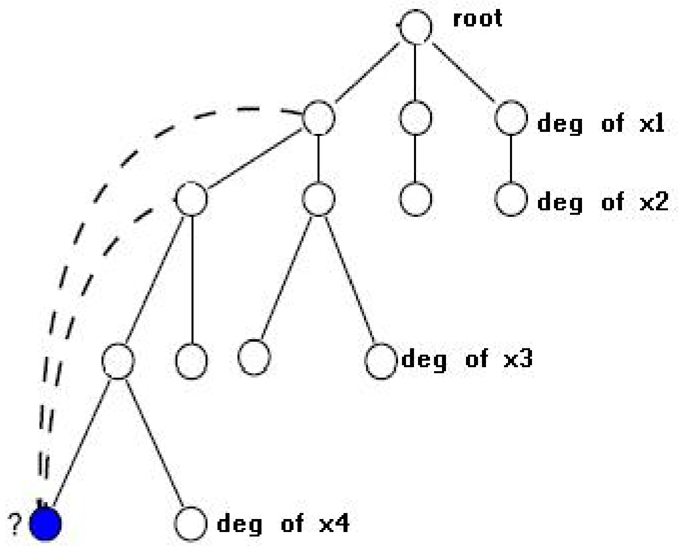
Recovery Algorithm



1. Start from root, pick all the nodes that have the degree of x1, to be root's children

2. Find neighbors of nodes in level 1, which have degree of x2, to be their corresponding children

Recovery Algorithm

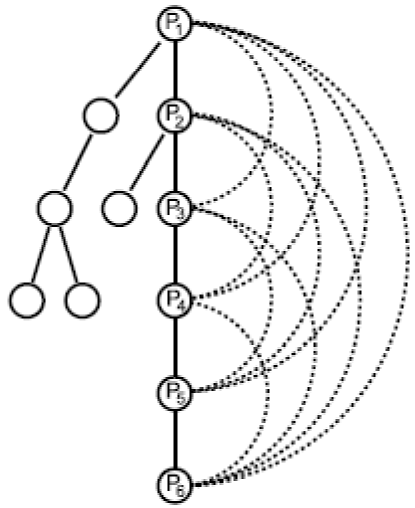


1. Start from root, pick all the nodes that have the degree of x1, to be root's children

2. Find neighbors of nodes in level 1, which have degree of x2, to be their corresponding children

3. Continue until level of k. If there's only one path from root, success; Otherwise, H is not unique. Try once more!

Recovery Runtime



Maximum degree of H - O(log n) keeps number of feasible paths relatively small since H is small

Analysis more complex, but in expectation, the number of feasible paths:

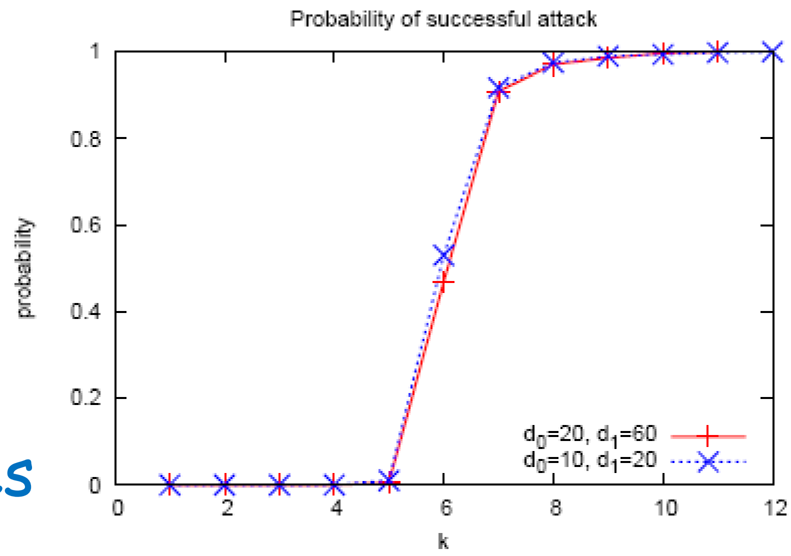
$$O(2^{O(\log \log n)^2})$$

Thus, total number of paths is only slightly superlinear, and so we can find H efficiently !

Experimental Results

- Simulated attack on LiveJournal – 4.4 million nodes, 77 million edges

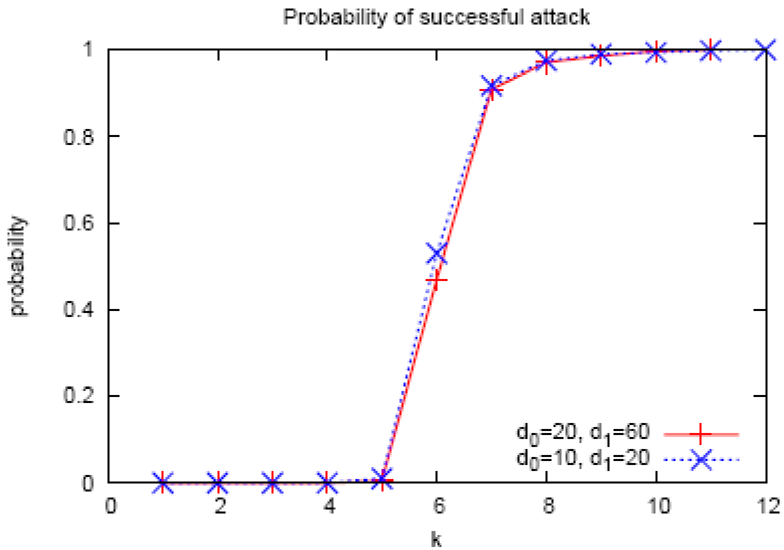
- Constructed H with degrees randomly selected in [10, 20] or [20, 60], while varying k



Experimental Results

-With 7 nodes, d in [20, 60], successful over 90%; compromises an average of 70 users

-Recovery typically takes less than a second ; size of search tree about 90,000



Comparison of Attack

Walk-Based Attack

- Adds $O(\log n)$ new nodes
- Can compromise $O(\log^2 n)$ nodes at most
- Simple to execute, difficult to detect

Cut-Based Attack (Using Gomory-Hu tree)

- Adds $O(\sqrt{\log n})$ new nodes – theoretical minimum for any attack of this style
- Attacks $O(\sqrt{\log n})$ nodes
- Easy for curator to detect on real data

- Both attacks work no matter how many people execute them

Conclusion

- Doesn’t apply to networks that are already public

- Cannot rely on anonymization to ensure privacy in Networks

- Further directions

1) Design a general interactive method whereby researchers may make queries about the network

2) Non-interactive methods where the released data is perturbed in such a way that it is still useful to researchers, but provably anonymized


Thank You...
You made my day !