Sybil attacks and tracking in the Tox network

zugz

Toxcon 2018

Front matter

Overview

1. The problem: metadata privacy
2. Tox's solution
3. Weaknesses
4. Alternatives

Metadata privacy

Article 12 of the Universal Declaration of Human Rights

“**No one shall be subjected to arbitrary interference with his privacy**, family, home **or correspondence**, nor to attacks upon his honour and reputation. Everyone has the right to the protection of the law against such interference or attacks.”

Fundamental aim of tox

Allow Alice and Bob to privately establish an authenticated encrypted direct network connection.

$$A \longleftrightarrow B$$

Fundamental aim of tox

Allow Alice and Bob to **privately** establish an authenticated encrypted direct network connection.

$$A \longleftrightarrow B$$

- "**Privately**":
  - no third party can observe that A and B establish a connection.

Centralised vs distributed

Centralised

Signal, Telegram, etc.

- Connections mediated by central servers.

Centralised vs distributed

Centralised

Signal, Telegram, etc.

- Connections mediated by central servers.
- So central servers have the opportunity to observe connections being established.

Centralised vs distributed

Centralised

Signal, Telegram, etc.

- Connections mediated by central servers.
- So central servers have the opportunity to observe connections being established.

Centralised vs distributed

Centralised

Signal, Telegram, etc.

- Connections mediated by central servers.
- So central servers have the opportunity to observe connections being established.

Distributed

Tox, Ring, etc.

- Role of central server is played by the whole network.

Centralised vs distributed

Centralised

Signal, Telegram, etc.

- Connections mediated by central servers.
- So central servers have the opportunity to observe connections being established.

Distributed

Tox, Ring, etc.

- Role of central server is played by the whole network.
- So **everyone** has the opportunity to observe connections being established!

Overview of tox

ID keys and Friends

Each tox user generates a long-term **ID keypair**

- The ID pubkey can be seriously public, e.g. published on user's website.
- Analagous to PGP pubkey.

ID keys and Friends

Each tox user generates a long-term **ID keypair**

- The ID pubkey can be seriously public, e.g. published on user's website.
- Analagous to PGP pubkey.

1. Given A's ID pubkey, any peer B may make a **Friend Request** to A.
2. If A accepts, A and B are then **Friends**.
3. Then whenever A and B are simultaneously online, they find each other via a **Friend Finding** process, and establish an authenticated encrypted direct network connection.

Privacy (first version)

Privacy requirement: no third party E can determine that A and B are Friends, nor that B is making a Friend Request to A.

Privacy (first version)

Privacy requirement: no third party E can determine that A and B are Friends, nor that B is making a Friend Request to A.

Important provisos:

1. We assume that the IP address of a peer does **not** identify that peer. E may well determine that whoever has A's IP address is Friends with whoever has B's IP address.
2. We assume E is not Friends with both A and B.

DHT

- Tox nodes form a (Kademlia-like) Distributed Hash Table.
- Consider the space of 32-byte bitstrings with distances between points given by the XOR metric

$$\mathrm{dist}(v, v') := v \oplus v' \in [0, \ldots, 2^{256} - 1].$$

- Each node generate a temporary **DHT keypair** and positions themselves in this space at their DHT pubkey.

DHT

- Tox nodes form a (Kademlia-like) Distributed Hash Table.
- Consider the space of 32-byte bitstrings with distances between points given by the XOR metric

$$\text{dist}(v, v') := v \oplus v' \in [0, \ldots, 2^{256} - 1].$$

- Each node generate a temporary **DHT keypair** and positions themselves in this space at their DHT pubkey.

- We can **search** in the DHT for a target bitstring by making requests to DHT nodes;
- DHT nodes respond with the nodes they know which are closest to the target;
- recursing, so we eventually find nodes with DHT pubkey close to the target.

DHT

- Tox nodes form a (Kademlia-like) Distributed Hash Table.
- Consider the space of 32-byte bitstrings with distances between points given by the XOR metric

$$\mathrm{dist}(v, v') := v \oplus v' \in [0, \ldots, 2^{256} - 1].$$

- Each node generate a temporary **DHT keypair** and positions themselves in this space at their DHT pubkey.

- We can **search** in the DHT for a target bitstring by making requests to DHT nodes;
- DHT nodes respond with the nodes they know which are closest to the target;
- recursing, so we eventually find nodes with DHT pubkey close to the target.

- The DHT uses UDP;
  - exists TCP relay system as a backup option.

Friend Finding

If A and B are Friends and both online, A connects to B as follows:

- Step 1: B sends their DHT pubkey to A via Onion (see below)
- Step 2: A searches for B's DHT pubkey in the DHT.
  - If no NAT prevents it, A finds and connects directly to B;
  - else, nodes adjacent to B in DHT mediate UDP holepunching;
  - while that fails, A also tries to connect to B via a TCP relay.
- Simultaneously, B tries to connect to A in the same way.
  - Only one direction needs to succeed.

Friend Finding

If A and B are Friends and both online, A connects to B as follows:

- Step 1: B sends their DHT pubkey to A via Onion (see below)
- Step 2: A searches for B's DHT pubkey in the DHT.
  - If no NAT prevents it, A finds and connects directly to B;
  - else, nodes adjacent to B in DHT mediate UDP holepunching;
  - while that fails, A also tries to connect to B via a TCP relay.
- Simultaneously, B tries to connect to A in the same way.
  - Only one direction needs to succeed.

Note: the fact that A and B are searching for each other is leaked to the DHT, with A and B identified by their IP addresses and DHT pubkeys.

- So: we must assume that IP addresses do not identify,
- and E must not be able to determine the ID pubkey associated to a given DHT pubkey.

Onion paths

- Want:
  - If B knows A's ID pubkey
  - then B can send a message to A, without revealing the fact to any E.
- This allows:
  - **Friend Finding**: can send DHT pubkey to Friend;
  - **Friend Requests**: can send FR to potential Friend.

Onion paths

- Want:
  - If B knows A's ID pubkey
  - then B can send a message to A, without revealing the fact to any E.
- This allows:
  - **Friend Finding**: can send DHT pubkey to Friend;
  - **Friend Requests**: can send FR to potential Friend.

Tox uses onion routing for this.

- An **onion path** is a series of UDP connections between DHT nodes
  $$A \longleftrightarrow O_1 \longleftrightarrow O_2 \longleftrightarrow O_3 \longleftrightarrow C.$$
- Using layered encryption, A and C can communicate along the onion path.
- Each node sees the IP addresses and DHT pubkeys of **only its immediate neighbours**.

Onion

- A establishes an onion path to a node C with DHT pubkey close to A's ID pubkey;

$$A \longleftrightarrow O_1 \longleftrightarrow O_2 \longleftrightarrow O_3 \longleftrightarrow C.$$

- B also establishes an onion path to C;
- B sends message to A via C and the onion paths.

$$A \leftarrow O_1 \leftarrow O_2 \leftarrow O_3 \leftarrow C \leftarrow O_3' \leftarrow O_2' \leftarrow O_1' \leftarrow B.$$

Onion

- A establishes an onion path to a node C with DHT pubkey close to A's ID pubkey;

$$A \longleftrightarrow O_1 \longleftrightarrow O_2 \longleftrightarrow O_3 \longleftrightarrow C.$$

- B also establishes an onion path to C;
- B sends message to A via C and the onion paths.

$$A \leftarrow O_1 \leftarrow O_2 \leftarrow O_3 \leftarrow C \leftarrow O_3' \leftarrow O_2' \leftarrow O_1' \leftarrow B.$$

- Intention:
  - no-one learns **both** A's ID pubkey **and** A's DHT pubkey;
  - only A learns B's ID pubkey.

Problems with the onion

Problem 1: Bug #1121

- B sends message to A via C and the onion paths.

  $$A \leftarrow O_1 \leftarrow O_2 \leftarrow O_3 \leftarrow C \leftarrow O_3' \leftarrow O_2' \leftarrow O_1' \leftarrow B.$$

  Intention: only A learns B's ID pubkey

Problem 1: Bug #1121

- B sends message to A via C and the onion paths.

  $$A \leftarrow O_1 \leftarrow O_2 \leftarrow O_3 \leftarrow C \leftarrow O_3' \leftarrow O_2' \leftarrow O_1' \leftarrow B.$$

  Intention: only A learns B's ID pubkey
- However, a vulnerability in the protocol lets C determine B's ID pubkey.

Problem 1: Bug #1121

- B sends message to A via C and the onion paths.

  $$A \leftarrow O_1 \leftarrow O_2 \leftarrow O_3 \leftarrow C \leftarrow O_3' \leftarrow O_2' \leftarrow O_1' \leftarrow B.$$

  Intention: only A learns B's ID pubkey
- However, a vulnerability in the protocol lets C determine B's ID pubkey.
- C also knows A's ID pubkey.

Problem 1: Bug #1121

- B sends message to A via C and the onion paths.

  $$A \leftarrow O_1 \leftarrow O_2 \leftarrow O_3 \leftarrow C \leftarrow O_3' \leftarrow O_2' \leftarrow O_1' \leftarrow B.$$

  Intention: only A learns B's ID pubkey
- However, a vulnerability in the protocol lets C determine B's ID pubkey.
- C also knows A's ID pubkey.
- Oops!

Problem 1: Bug #1121

- B sends message to A via C and the onion paths.

  $$A \leftarrow O_1 \leftarrow O_2 \leftarrow O_3 \leftarrow C \leftarrow O_3' \leftarrow O_2' \leftarrow O_1' \leftarrow B.$$

  Intention: only A learns B's ID pubkey
- However, a vulnerability in the protocol lets C determine B's ID pubkey.
- C also knows A's ID pubkey.
- Oops!

Problem 1: Bug #1121

- B sends message to A via C and the onion paths.

  $$A \leftarrow O_1 \leftarrow O_2 \leftarrow O_3 \leftarrow C \leftarrow O_3' \leftarrow O_2' \leftarrow O_1' \leftarrow B.$$

  Intention: only A learns B's ID pubkey
- However, a vulnerability in the protocol lets C determine B's ID pubkey.
- C also knows A's ID pubkey.
- Oops!

Details:

- B sends their ID pubkey encrypted to a *data pubkey* set by A and communicated to B by C.

Problem 1: Bug #1121

- B sends message to A via C and the onion paths.

  $$A \leftarrow O_1 \leftarrow O_2 \leftarrow O_3 \leftarrow C \leftarrow O_3' \leftarrow O_2' \leftarrow O_1' \leftarrow B.$$

  Intention: only A learns B's ID pubkey
- However, a vulnerability in the protocol lets C determine B's ID pubkey.
- C also knows A's ID pubkey.
- Oops!

Details:

- B sends their ID pubkey encrypted to a *data pubkey* set by A and communicated to B by C.
- But C can just substitute the data pubkey supplied by A for their own.

Problem 1: Bug #1121

- B sends message to A via C and the onion paths.

$$A \leftarrow O_1 \leftarrow O_2 \leftarrow O_3 \leftarrow C \leftarrow O_3' \leftarrow O_2' \leftarrow O_1' \leftarrow B.$$

  Intention: only A learns B's ID pubkey
- However, a vulnerability in the protocol lets C determine B's ID pubkey.
- C also knows A's ID pubkey.
- Oops!

Details:

- B sends their ID pubkey encrypted to a *data pubkey* set by A and communicated to B by C.
- But C can just substitute the data pubkey supplied by A for their own.
- Oops.

Problem 1: Bug #1121

- B sends message to A via C and the onion paths.

$$A \leftarrow O_1 \leftarrow O_2 \leftarrow O_3 \leftarrow C \leftarrow O_3' \leftarrow O_2' \leftarrow O_1' \leftarrow B.$$

  Intention: only A learns B's ID pubkey
- However, a vulnerability in the protocol lets C determine B's ID pubkey.
- C also knows A's ID pubkey.
- Oops!

Details:

- B sends their ID pubkey encrypted to a *data pubkey* set by A and communicated to B by C.
- But C can just substitute the data pubkey supplied by A for their own.
- Oops.
- Can't fix without breaking protocol compatibility.

Problem 2: Paths

$$A \longleftrightarrow O_1 \longleftrightarrow O_2 \longleftrightarrow O_3 \longleftrightarrow C$$

- Intention: no-one learns **both** A's ID pubkey **and** A's DHT pubkey.

Problem 2: Paths

$$A \longleftrightarrow O_1 \longleftrightarrow O_2 \longleftrightarrow O_3 \longleftrightarrow C$$

- Intention: no-one learns **both** A's ID pubkey **and** A's DHT pubkey.
- O_1 learns the DHT pubkey, and C learns the ID pubkey.

Problem 2: Paths

$$A \longleftrightarrow O_1 \longleftrightarrow O_2 \longleftrightarrow O_3 \longleftrightarrow C$$

- Intention: no-one learns **both** A's ID pubkey **and** A's DHT pubkey.
- O_1 learns the DHT pubkey, and C learns the ID pubkey.
- So A needs to avoid building **compromised** onion paths in which O_1 and C are controlled by the same entity.

Problem 2: Paths

$$A \longleftrightarrow O_1 \longleftrightarrow O_2 \longleftrightarrow O_3 \longleftrightarrow C$$

- Intention: no-one learns **both** A's ID pubkey **and** A's DHT pubkey.
- O_1 learns the DHT pubkey, and C learns the ID pubkey.
- So A needs to avoid building **compromised** onion paths in which O_1 and C are controlled by the same entity.
- With the current choice of path nodes, paths are often compromised, because often $O_1 = C$!

Problem 2: Paths

$$A \longleftrightarrow O_1 \longleftrightarrow O_2 \longleftrightarrow O_3 \longleftrightarrow C$$

- Intention: no-one learns **both** A's ID pubkey **and** A's DHT pubkey.
- O_1 learns the DHT pubkey, and C learns the ID pubkey.
- So A needs to avoid building **compromised** onion paths in which O_1 and C are controlled by the same entity.
- With the current choice of path nodes, paths are often compromised, because often $O_1 = C$!
- Can try to choose path nodes more cleverly (#596), but…

Sybil and Eclipse attacks

- Attacker aims to ensure
  1. large proportion of the DHT nodes we see are controlled by the attacker (call such nodes **malicious**, and the other nodes **honest**);
  2. we can't differentiate malicious nodes from honest nodes.
- Then any onion path we build is likely to be compromised.
  - (Thanks to nazar_pc for convincing me to take this seriously).

Sybil and Eclipse attacks

- Attacker aims to ensure
  1. large proportion of the DHT nodes we see are controlled by the attacker (call such nodes **malicious**, and the other nodes **honest**);
  2. we can't differentiate malicious nodes from honest nodes.
- Then any onion path we build is likely to be compromised.
  - (Thanks to nazar_pc for convincing me to take this seriously).

- **Sybil attack** (Global):
  - Attacker simply runs enough DHT nodes that a large proportion of the whole DHT network is malicious.
  - e.g. the current Tox DHT has <10000 nodes, so running a few thousand nodes is enough for a strong Sybil attack.

Sybil and Eclipse attacks

- Attacker aims to ensure
  1. large proportion of the DHT nodes we see are controlled by the attacker (call such nodes **malicious**, and the other nodes **honest**);
  2. we can't differentiate malicious nodes from honest nodes.
- Then any onion path we build is likely to be compromised.
  - (Thanks to nazar_pc for convincing me to take this seriously).
- **Sybil attack** (Global):
  - Attacker simply runs enough DHT nodes that a large proportion of the whole DHT network is malicious.
  - e.g. the current Tox DHT has $<$10000 nodes, so running a few thousand nodes is enough for a strong Sybil attack.
- **Eclipse attack** (Targeted):
  - Poison target's view of the DHT by responding with malicious nodes to any requests ("eclipsing" honest nodes).
    - (can mint new nodes on demand to fit the requests.)
  - Malicious nodes rapidly dominate target's view.

A cat picture

A cat picture

A cat picture

Zoff suggests I put a cat picture in my talk to stop it being too dry.

A cat picture

Zoff suggests I put a cat picture in my talk to stop it being too dry.

Here is a cat picture.

A cat picture

Zoff suggests I put a cat picture in my talk to stop it being too dry.

Here is a cat picture.

It is released under the Creative Commons Attribution-Sharealike License version 3.0. The photo is by Ryan Finnie.

A cat picture

Zoff suggests I put a cat picture in my talk to stop it being too dry.

Here is a cat picture.

It is released under the Creative Commons Attribution-Sharealike License version 3.0. The photo is by Ryan Finnie.

Defences against Sybil attacks

...and why they're inappropriate/insufficient for Tox

Adapted from G. Urdaneta, G. Pierre, M. Steen "*A survey of DHT security techniques*" ACM Comput. Surv. 2011.

Defences against Sybil attacks

. . . and why they're inappropriate/insufficient for Tox

Adapted from G. Urdaneta, G. Pierre, M. Steen "*A survey of DHT security techniques*" ACM Comput. Surv. 2011.

1. Central register of trusted nodes.

Defences against Sybil attacks

. . . and why they're inappropriate/insufficient for Tox

Adapted from G. Urdaneta, G. Pierre, M. Steen "*A survey of DHT security techniques*" ACM Comput. Surv. 2011.

1. Central register of trusted nodes.
    - Incompatible with distributedness and privacy.

Defences against Sybil attacks

…and why they're inappropriate/insufficient for Tox

Adapted from G. Urdaneta, G. Pierre, M. Steen "*A survey of DHT security techniques*" ACM Comput. Surv. 2011.

1. Central register of trusted nodes.
    - Incompatible with distributedness and privacy.
2. Assume attacker controls few IP addresses / physical network locations.

Defences against Sybil attacks

…and why they're inappropriate/insufficient for Tox

Adapted from G. Urdaneta, G. Pierre, M. Steen "*A survey of DHT security techniques*" ACM Comput. Surv. 2011.

1. Central register of trusted nodes.
   - Incompatible with distributedness and privacy.
2. Assume attacker controls few IP addresses / physical network locations.
   - Fails against a botnet.

Defences against Sybil attacks

…and why they're inappropriate/insufficient for Tox

Adapted from G. Urdaneta, G. Pierre, M. Steen "*A survey of DHT security techniques*" ACM Comput. Surv. 2011.

1. Central register of trusted nodes.
   - Incompatible with distributedness and privacy.
2. Assume attacker controls few IP addresses / physical network locations.
   - Fails against a botnet.
3. Computational puzzles ("Proof of Work").

Defences against Sybil attacks

…and why they're inappropriate/insufficient for Tox

Adapted from G. Urdaneta, G. Pierre, M. Steen "*A survey of DHT security techniques*" ACM Comput. Surv. 2011.

1. Central register of trusted nodes.
   - Incompatible with distributedness and privacy.
2. Assume attacker controls few IP addresses / physical network locations.
   - Fails against a botnet.
3. Computational puzzles ("Proof of Work").
   - Forces honest nodes to waste CPU time;

Defences against Sybil attacks

. . . and why they're inappropriate/insufficient for Tox

Adapted from G. Urdaneta, G. Pierre, M. Steen "*A survey of DHT security techniques*" ACM Comput. Surv. 2011.

1. Central register of trusted nodes.
   - Incompatible with distributedness and privacy.
2. Assume attacker controls few IP addresses / physical network locations.
   - Fails against a botnet.
3. Computational puzzles ("Proof of Work").
   - Forces honest nodes to waste CPU time;
   - Fails against a sufficiently resourceful adversary.

Defences against Sybil attacks

…and why they're inappropriate/insufficient for Tox

Adapted from G. Urdaneta, G. Pierre, M. Steen "*A survey of DHT security techniques*" ACM Comput. Surv. 2011.

1. Central register of trusted nodes.
   - Incompatible with distributedness and privacy.
2. Assume attacker controls few IP addresses / physical network locations.
   - Fails against a botnet.
3. Computational puzzles ("Proof of Work").
   - Forces honest nodes to waste CPU time;
   - Fails against a sufficiently resourceful adversary.
4. Use a cryptocurrency.

Defences against Sybil attacks

. . . and why they're inappropriate/insufficient for Tox

Adapted from G. Urdaneta, G. Pierre, M. Steen "*A survey of DHT security techniques*" ACM Comput. Surv. 2011.

1. Central register of trusted nodes.
    - Incompatible with distributedness and privacy.
2. Assume attacker controls few IP addresses / physical network locations.
    - Fails against a botnet.
3. Computational puzzles ("Proof of Work").
    - Forces honest nodes to waste CPU time;
    - Fails against a sufficiently resourceful adversary.
4. Use a cryptocurrency.
    - Costs honest nodes money.

Defences against Sybil attacks

...and why they're inappropriate/insufficient for Tox

Adapted from G. Urdaneta, G. Pierre, M. Steen "*A survey of DHT security techniques*" ACM Comput. Surv. 2011.

1. Central register of trusted nodes.
   - Incompatible with distributedness and privacy.
2. Assume attacker controls few IP addresses / physical network locations.
   - Fails against a botnet.
3. Computational puzzles ("Proof of Work").
   - Forces honest nodes to waste CPU time;
   - Fails against a sufficiently resourceful adversary.
4. Use a cryptocurrency.
   - Costs honest nodes money.
5. Social graph techniques; e.g. SybilGuard, SybilLimit.

Defences against Sybil attacks

. . . and why they're inappropriate/insufficient for Tox

Adapted from G. Urdaneta, G. Pierre, M. Steen "*A survey of DHT security techniques*" ACM Comput. Surv. 2011.

1. Central register of trusted nodes.
   - Incompatible with distributedness and privacy.
2. Assume attacker controls few IP addresses / physical network locations.
   - Fails against a botnet.
3. Computational puzzles ("Proof of Work").
   - Forces honest nodes to waste CPU time;
   - Fails against a sufficiently resourceful adversary.
4. Use a cryptocurrency.
   - Costs honest nodes money.
5. Social graph techniques; e.g. SybilGuard, SybilLimit.
   - See next slide.

SybilGuard

- H. Yu, M. Kaminsky, P. Gibbons, A. Flaxman "*Sybilguard: defending against Sybil attacks via social networks*" ACM SIGCOMM 2006.

SybilGuard

- H. Yu, M. Kaminsky, P. Gibbons, A. Flaxman "*Sybilguard: defending against Sybil attacks via social networks*" ACM SIGCOMM 2006.
- *Idea*: in the Friend graph, the malicious Sybil subgraph will be connected by few edges to the honest subgraph. Can use this to identify honest nodes.

SybilGuard

- H. Yu, M. Kaminsky, P. Gibbons, A. Flaxman “*Sybilguard: defending against Sybil attacks via social networks*” ACM SIGCOMM 2006.
- *Idea*: in the Friend graph, the malicious Sybil subgraph will be connected by few edges to the honest subgraph. Can use this to identify honest nodes.
- **However**: SybilGuard requires the social graph to be public!

SybilGuard

- H. Yu, M. Kaminsky, P. Gibbons, A. Flaxman "*Sybilguard: defending against Sybil attacks via social networks*" ACM SIGCOMM 2006.
- *Idea*: in the Friend graph, the malicious Sybil subgraph will be connected by few edges to the honest subgraph. Can use this to identify honest nodes.
- **However**: SybilGuard requires the social graph to be public!
- Can adapt the system to be privacy-aware; but:

SybilGuard

- H. Yu, M. Kaminsky, P. Gibbons, A. Flaxman "*Sybilguard: defending against Sybil attacks via social networks*" ACM SIGCOMM 2006.
- *Idea*: in the Friend graph, the malicious Sybil subgraph will be connected by few edges to the honest subgraph. Can use this to identify honest nodes.
- **However**: SybilGuard requires the social graph to be public!
- Can adapt the system to be privacy-aware; but:
  - nodes need to have **some** stable public ID to be checked for honesty, with link between this ID and current DHT identity being public.

SybilGuard

- H. Yu, M. Kaminsky, P. Gibbons, A. Flaxman "*Sybilguard: defending against Sybil attacks via social networks*" ACM SIGCOMM 2006.
- *Idea*: in the Friend graph, the malicious Sybil subgraph will be connected by few edges to the honest subgraph. Can use this to identify honest nodes.
- **However**: SybilGuard requires the social graph to be public!
- Can adapt the system to be privacy-aware; but:
  - nodes need to have **some** stable public ID to be checked for honesty, with link between this ID and current DHT identity being public.
  - This itself is a privacy problem!

SybilGuard

- H. Yu, M. Kaminsky, P. Gibbons, A. Flaxman "*Sybilguard: defending against Sybil attacks via social networks*" ACM SIGCOMM 2006.
- *Idea*: in the Friend graph, the malicious Sybil subgraph will be connected by few edges to the honest subgraph. Can use this to identify honest nodes.
- **However**: SybilGuard requires the social graph to be public!
- Can adapt the system to be privacy-aware; but:
  - nodes need to have **some** stable public ID to be checked for honesty, with link between this ID and current DHT identity being public.
  - This itself is a privacy problem!
  - Indeed, Friend relation between DHT pubkeys is leaked,

SybilGuard

- H. Yu, M. Kaminsky, P. Gibbons, A. Flaxman "*Sybilguard: defending against Sybil attacks via social networks*" ACM SIGCOMM 2006.
- *Idea*: in the Friend graph, the malicious Sybil subgraph will be connected by few edges to the honest subgraph. Can use this to identify honest nodes.
- **However**: SybilGuard requires the social graph to be public!
- Can adapt the system to be privacy-aware; but:
  - nodes need to have **some** stable public ID to be checked for honesty, with link between this ID and current DHT identity being public.
  - This itself is a privacy problem!
  - Indeed, Friend relation between DHT pubkeys is leaked,
  - hence Friend relation between these stable IDs leaked;

SybilGuard

- H. Yu, M. Kaminsky, P. Gibbons, A. Flaxman "*Sybilguard: defending against Sybil attacks via social networks*" ACM SIGCOMM 2006.
- *Idea*: in the Friend graph, the malicious Sybil subgraph will be connected by few edges to the honest subgraph. Can use this to identify honest nodes.
- **However**: SybilGuard requires the social graph to be public!
- Can adapt the system to be privacy-aware; but:
  - nodes need to have **some** stable public ID to be checked for honesty, with link between this ID and current DHT identity being public.
  - This itself is a privacy problem!
  - Indeed, Friend relation between DHT pubkeys is leaked,
  - hence Friend relation between these stable IDs leaked;
  - then attacker can use IP-based techniques to link these IDs to real-life identities,

SybilGuard

- H. Yu, M. Kaminsky, P. Gibbons, A. Flaxman "*Sybilguard: defending against Sybil attacks via social networks*" ACM SIGCOMM 2006.
- *Idea*: in the Friend graph, the malicious Sybil subgraph will be connected by few edges to the honest subgraph. Can use this to identify honest nodes.
- **However**: SybilGuard requires the social graph to be public!
- Can adapt the system to be privacy-aware; but:
  - nodes need to have **some** stable public ID to be checked for honesty, with link between this ID and current DHT identity being public.
  - This itself is a privacy problem!
  - Indeed, Friend relation between DHT pubkeys is leaked,
  - hence Friend relation between these stable IDs leaked;
  - then attacker can use IP-based techniques to link these IDs to real-life identities,
  - so full Friend graph is essentially leaked.

Accepting Sybil

Accepting Sybil

- **Assume** the nodes in the DHT views of A and B are controlled by E.

Accepting Sybil

- **Assume** the nodes in the DHT views of A and B are controlled by E.
- Then **Private Friend Requests are infeasible**.

Accepting Sybil

- **Assume** the nodes in the DHT views of A and B are controlled by E.
- Then **Private Friend Requests are infeasible**.
- Indeed: suppose B wants to send a message to A via E's nodes, using no secrets shared between A and B.

Accepting Sybil

- **Assume** the nodes in the DHT views of A and B are controlled by E.
- Then **Private Friend Requests are infeasible**.
- Indeed: suppose B wants to send a message to A via E's nodes, using no secrets shared between A and B.
- Then B must essentially **broadcast** the message (encrypted to A) to the **whole** network.

Accepting Sybil

- **Assume** the nodes in the DHT views of A and B are controlled by E.
- Then **Private Friend Requests are infeasible**.
- Indeed: suppose B wants to send a message to A via E's nodes, using no secrets shared between A and B.
- Then B must essentially **broadcast** the message (encrypted to A) to the **whole** network.
- Rough argument:

Accepting Sybil

- **Assume** the nodes in the DHT views of A and B are controlled by E.
- Then **Private Friend Requests are infeasible**.
- Indeed: suppose B wants to send a message to A via E's nodes, using no secrets shared between A and B.
- Then B must essentially **broadcast** the message (encrypted to A) to the **whole** network.
- Rough argument:
  - Otherwise, only some nodes interact with E so as to obtain the message;

Accepting Sybil

- **Assume** the nodes in the DHT views of A and B are controlled by E.
- Then **Private Friend Requests are infeasible**.
- Indeed: suppose B wants to send a message to A via E's nodes, using no secrets shared between A and B.
- Then B must essentially **broadcast** the message (encrypted to A) to the **whole** network.
- Rough argument:
  - Otherwise, only some nodes interact with E so as to obtain the message;
  - but how a node interacts is based on their ID pubkey,

Accepting Sybil

- **Assume** the nodes in the DHT views of A and B are controlled by E.
- Then **Private Friend Requests are infeasible**.
- Indeed: suppose B wants to send a message to A via E's nodes, using no secrets shared between A and B.
- Then B must essentially **broadcast** the message (encrypted to A) to the **whole** network.
- Rough argument:
  - Otherwise, only some nodes interact with E so as to obtain the message;
  - but how a node interacts is based on their ID pubkey,
  - so E obtains some information on A's ID pubkey.

Accepting Sybil

- **Assume** the nodes in the DHT views of A and B are controlled by E.
- Then **Private Friend Requests are infeasible**.
- Indeed: suppose B wants to send a message to A via E's nodes, using no secrets shared between A and B.
- Then B must essentially **broadcast** the message (encrypted to A) to the **whole** network.
- Rough argument:
  - Otherwise, only some nodes interact with E so as to obtain the message;
  - but how a node interacts is based on their ID pubkey,
  - so E obtains some information on A's ID pubkey.
  - B might know no honest targets other than A, nor how many there are, so can't control how much information E obtains this way.

Private Friend Finding

However:

Claim

- If A and B co-operate and use a shared secret, they **can** privately and efficiently exchange messages via E's nodes.
- **Private Friend Finding is feasible**.

Life without FRs

Making friends by request

Currently, if A and B are real-life friends who wish to become Tox Friends, they proceed as follows.

1. Via some (hopefully authenticated) channel, A sends their ID pubkey to B, (along with a “nospam” password).
2. B makes an FR using A’s ID pubkey (and nospam), along with an optional message (“Bob here! Honest!”).
3. A is shown B’s ID pubkey and the message, and decides whether or not to accept the FR.

Making friends by request

Currently, if A and B are real-life friends who wish to become Tox Friends, they proceed as follows.

1. Via some (hopefully authenticated) channel, A sends their ID pubkey to B, (along with a "nospam" password).
2. B makes an FR using A's ID pubkey (and nospam), along with an optional message ("Bob here! Honest!").
3. A is shown B's ID pubkey and the message, and decides whether or not to accept the FR.

- Should she? How does she know this isn't a MitM?

Making friends by request

Currently, if A and B are real-life friends who wish to become Tox Friends, they proceed as follows.

1. Via some (hopefully authenticated) channel, A sends their ID pubkey to B, (along with a “nospam” password).
2. B makes an FR using A’s ID pubkey (and nospam), along with an optional message (“Bob here! Honest!”).
3. A is shown B’s ID pubkey and the message, and decides whether or not to accept the FR.

- Should she? How does she know this isn’t a MitM?
- The ID pubkey and nospam may well be publically known, and relying on the message is very weak security.

Making friends by request

Currently, if A and B are real-life friends who wish to become Tox Friends, they proceed as follows.

1. Via some (hopefully authenticated) channel, A sends their ID pubkey to B, (along with a “nospam” password).
2. B makes an FR using A’s ID pubkey (and nospam), along with an optional message (“Bob here! Honest!”).
3. A is shown B’s ID pubkey and the message, and decides whether or not to accept the FR.

- Should she? How does she know this isn’t a MitM?
- The ID pubkey and nospam may well be publically known, and relying on the message is very weak security.
- So she should ask B to send B’s ID pubkey via their separate channel, and check it agrees with the pubkey in the FR.

Making friends by request

Currently, if A and B are real-life friends who wish to become Tox Friends, they proceed as follows.

1. Via some (hopefully authenticated) channel, A sends their ID pubkey to B, (along with a “nospam” password).
2. B makes an FR using A’s ID pubkey (and nospam), along with an optional message (“Bob here! Honest!”).
3. A is shown B’s ID pubkey and the message, and decides whether or not to accept the FR.

- Should she? How does she know this isn’t a MitM?
- The ID pubkey and nospam may well be publically known, and relying on the message is very weak security.
- So she should ask B to send B’s ID pubkey via their separate channel, and check it agrees with the pubkey in the FR.
- But then, why have the FR at all?

Making friends without FRs

Symmetric approach:

1. Via some (hopefully authenticated) channel, A and B exchange their ID pubkeys.
2. Each adds the other in their Tox client.
3. They connect by Friend Finding.

No opportunity for MitM. No nospam required.

Making friends without FRs

Symmetric approach:

1. Via some (hopefully authenticated) channel, A and B exchange their ID pubkeys.
2. Each adds the other in their Tox client.
3. They connect by Friend Finding.

No opportunity for MitM. No nospam required.

However, this doesn't cover one important use case:

- Sometimes, we want to be "**promiscuous**" - accept as a Friend anyone who tries to connect to us.
- With FRs, this just means accepting every FR.
- e.g. most bots function this way.
- Without FRs, we need some other mechanism to permit promiscuity.

Proposal for private Friend Finding

PFFP

Claim

Exists feasible system with the *Private Friend Finding Property*.

PFFP

Claim

Exists feasible system with the *Private Friend Finding Property*.

Definition

Suppose A connects to some B_1 via the system, then on an *independent occasion* connects to some B_2.

The system has the **Private Friend Finding Property** (**PFFP**) if no *independent third party* E can distinguish this pair of connection events from a random pair of connection events.

- **"independent occasion"**: A uses a fresh independent IP address on the second occasion, and the occasions are sufficiently separated in time.
- **"independent third party"**: E not a Friend of A, and $E \neq B_i$.
- Motivation: Without PFFP, E can track users and reconstruct the friend graph.

Achieving PFFP: basic idea

Suggested by Grayhatter.
https://wiki.cmdline.org/doku.php?id=dht:new_friend_finding

Setup: A and B know each others' ID pubkeys, and actively co-operate to exchange messages.

Achieving PFFP: basic idea

Suggested by Grayhatter.
https://wiki.cmdline.org/doku.php?id=dht:new_friend_finding

Setup: *A* and *B* know each others' ID pubkeys, and actively co-operate to exchange messages.

1. Using shared secrets (e.g. the shared key derived from their ID keypairs), *A* and *B* each calculate a shared *rendezvous location* on the DHT.
2. Both connect directly to the DHT nodes nearest the rendezvous location.
3. They exchange messages via these rendezvous nodes, encrypted and authenticated with their ID keys.

Achieving PFFP: basic idea

Suggested by Grayhatter.
https://wiki.cmdline.org/doku.php?id=dht:new_friend_finding

Setup: *A* and *B* know each others' ID pubkeys, and actively co-operate to exchange messages.

1. Using shared secrets (e.g. the shared key derived from their ID keypairs), *A* and *B* each calculate a shared *rendezvous location* on the DHT.
2. Both connect directly to the DHT nodes nearest the rendezvous location.
3. They exchange messages via these rendezvous nodes, encrypted and authenticated with their ID keys.

- Complications:
  1. For PFFP, the location must vary with time.
  2. For scalability, we don't actually want separate rendezvous locations for each pair of friends.
  3. Need to permit promiscuity.
  4. Need a mechanism for using the system in TCP-only mode.

Complication 1: Time-varying rendezvous

- Use rounded unix time as an input to rendezvous calculation, say rounded to the hour.
- To deal with inaccurate clocks, use both locations near the cut-off.
- To prevent fingerprinting, add random error to clock on startup.
- Details:
  - Tweakable constants $E < M < P$;
  - Suggested values: $E := 300; M := 900; P := 3600$.
  - $e \sim \mathrm{Unif}([-E, E])$;
  - $\mathrm{roundedTime}(t, \epsilon) := (t + e + (-1)^{\epsilon} M/2)/P$;
  - Use both $\mathrm{roundedTime}(\mathrm{unixTime}, 0)$ and $\mathrm{roundedTime}(\mathrm{unixTime}, 1)$ as inputs when generating rendezvous.

System clock

- This relies on system clock being approximately correct.
- Friend finding silently fails if it's too far out.
  - (if the difference between our error-adjusted clock and that of our friend is dt, we generate a common rendezvous $\max(0, \min(1, (\mathrm{dt} - M)/P))$ of the time)
- Clock could be used to fingerprint if too far out (compared to E).
- Maybe: clients should check with an NTP server at startup, and warn the user if the clock appears to be significantly wrong.

Complication 2: Linearly many rendezvous

- A **rendezvous key** (**rdvkey**) is a 32-byte bitstring.
- For each friend, we keep track of the rdvkey they have for us.
- For a new friend, the "*initial*" rdvkey is the shared symmetric encryption key generated from our ID keys.
- We separately generate a single "*common*" rdvkey, which is a signing key.
- We send our common rdvkey to friends on making friend connections.
- For each rdvkey held by a friend, and each $\epsilon \in \{0, 1\}$, we **announce** to
  - $\mathrm{announceLocation}(\mathrm{rdv}, \mathrm{ourPubkey}, \epsilon) := SHA256(\mathrm{rdv} \mathbin{++} \mathrm{roundedTime}(\mathrm{unixTime}, \epsilon) \mathbin{++} \mathrm{ourPubkey})$
- i.e. we store our DHT pubkey there, signed/encrypted with the rdvkey.
- Friends **search** at these same locations to obtain our DHT pubkey.

Wrinkle: resetting the common key

- Only friends should have our common key.
- So if we delete a friend, we must replace the common key with a freshly generated key.
- We must then revert to the per-friend initial rdvkeys for our offline friends, and tell them the new key when we next connect to them.
- Because we never know whether a common key we are searching at is valid, we must also always search (possibly less intensively) at the initial key.

Complication 3: Promiscuity

If we want to be promiscuous:

- We generate a signing key, and distribute it as a "*public*" rdvkey.
- We announce with the public rdvkey.
- A potential user of the bot enters into their client this public rdvkey along with our ID pubkey, and they search for us as with a common rdvkey.
- Once they find our DHT pubkey, they try to connect to us as usual.
- They will get a DHTpk packet or TCP OOB net crypto handshake packet to us, which includes their ID pubkey.
- We then add them as a Friend.

This is flexible for different kinds of promiscuity. It could also be used as e.g. an "invite code" intended for limited distribution.

Simplifying promiscuity

We could use our ID pubkey as our public rdvkey.

This simplifies promiscuity - on first adding a friend, we can try both at the initial rdvkey and at their public rdvkey, and have a toggleable “promiscuous mode” (intended for bots and those who really don’t care about the privacy consequences) in which we announce at our public rdvkey.

Details: first generate an EdDSA signing keypair and then extract the Curve25519 ID keypair using the homomorphic birationality of the curves, crypto_sign_ed25519_sk_to_curve25519 in libsodium. Annoyingly, libsodium doesn’t provide the reverse function for the public keys (“crypto_sign_curve25519_pk_to_ed25519”), so we’d have to implement it ourselves (theoretically this is just a couple of lines of modular arithmetic), or get libsodium to.

Complication 4: TCP

- There is an Onion TCP packet accepted by TCP servers, allowing TCP-only clients to use the onion.
- An analogous mechanism would be needed to have TCP servers relay rendezvous packets to DHT nodes.
- Bonus: this could also be used by NGC where it currently wastefully uses the onion.

TODO

- Protocol details. Model on onion announce.
- Add mechanism for checking our announce nodes accept incoming connections?
- Handling transition from onion.
- Currently, we save known DHT nodes between sessions. This invalidates PFFP, and can be used to track us across sessions if our DHT view is poisoned.

Bonus: DoS resistance

- With the onion, your announce location is your ID pubkey, so anyone knowing your ID pubkey can prevent friends from finding you by filling your announce neighbourhood with malicious nodes.
- With the rendezvous system, only your friends know your announce locations.
- (But the public rdvkeys for the promiscuous case are still DoSsable)

Bonus: traffic

- With the onion, we have to maintain UDP connections with our announce nodes, so we must announce frequently and be ready to handle incoming packets.
- With the rendezvous system, we need only announce frequently enough to keep up with DHT churn, and need keep no connections open.
- Moreover, each packet is much lighter and doesn't need to be bounced around.
- So traffic requirements are **much** less than with the onion.
  - (on the order of a factor of 10 for announcing)
- Moreover, if we're not part of the DHT, we can sleep between announces/searches, conserving battery.