3. Security & Privacy for P2P

- Security & Privacy for P2P
    - Motivation
    - Attacks
    - Privacy Techniques
        - Crowds
        - Onion routing
            - Tarzan
    - Kademlia
    - S/Kademlia

1. Motivation

2. Attacks

There are many ways to attack distributed systems, and security is truly an important part to consider when one seeks to implement different protocols.

- `DDOS` people could try to overload the server via a Distributed Denial of Service attack.
- `Malicious Peer` If a *malicious peer* joins the network, he could reroute traffic and such
- `Sybil` If many malicious peers join, we have a sybil attack, and they could do all kinds of thing
- `Eclipse` Lastly a Eclipse attack would be to isolate a peer

3. Privacy Techniques

Now, privacy is a topic i really care about. It is not because I have something to hide, but because I am human. The fact that anybody knows everything I do, frightens me. Privacy, the choice of who gets to know what about me, is mine and only mine.

3.1. Crowds

When ever I send a request to a website, at most times, some third party websites gets notice of this and tracks me, which with enough data, will be able to create a profile on me and my behaviour.

Crowds is a system there can help to prevent this, by obscuration and reroute the way I load the page. Having joined a crowed, I ask a random member of it, perhaps Carol if he would get me the resource, perhaps he will or perhaps he pass along the request to another, and so on.

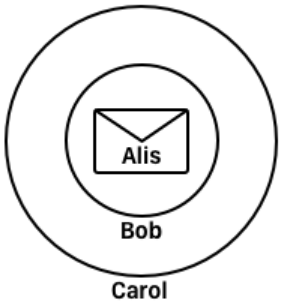
- ***Crowds***
- *Somebody from a crowd makes the request*

But this is rather simple, and now every one in the chain of the request know that I made it! We need something better.

3.2. Onion routing

Now what can we do to hide some information from others? Well we can encrypt it! But Magnus .. what does that have to do with onions? You ask...

Well, image that we have a system with public/private key pairs. Then I take my message to Alis and encrypt it with her public key. The encrypted message I take and encrypt with bobs key! And that encrypted message of an encrypted message I take and encrypt with Carol key! This is some true inception of encryptions.



But now, I send the message to carol, who can only decrypt it and pass it along to Bob, who can only decrypt it and see that now its Alis' turn, who can finally decrypt me real message.

Now, only Carol know who was the original sender and only bob know the original receiver; but neither knows the route of the message or that it was two endpoints.

Before, networks as these relied on a cloud of known peers (called mixers), which was then easy so block - and that is kind of a problem.

Enter, Tarzan!

3.2.1. Tarzan

The tarzan system is a decentralized, distributed onion system that enables client applications to seamlessly direct traffic through an anonymous network at the transport layer.

Being that it is a P2P network, it can contain thousand of peers, making it practically impossible to block them all; thus that wont be a problem any more.

- ***Tarzan***
- `P2P, all are mixer`
- `Robust` *against malicious peers*
- `Anonym` *tarzan azures anonymity*
- `Works on IP level`
- ***Joining***
- `Ask for peers` *retrieves a list of peers from a known peer*
- `Validate peers` *by pinging and get their peers*
- `Repeat` *repeated until the peer is satisfied*
- *Profit*

From these peers, the new node select a set to act as mimics, which it will exchange a constant rate of encrypted cover traffic of fixed size packets with.

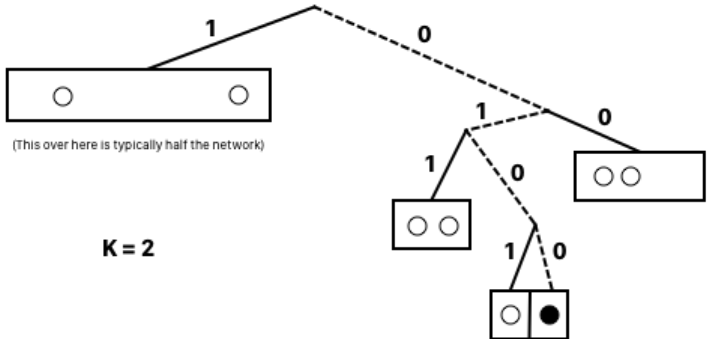
Now, whenever a node needs to send actual data, it can be interwoven within the cover traffic, without an observer detecting where a message originates. Where encryption layers are used as before.

Nodes store a record for the return to, so when a reply comes, they each rewrap it with their own private key; and since the original node is the only one that knows the path he is also the only one who knows which public keys to use to decrypt the returning message.

Is this way, Tarzan makes it hard to block all mixers and do to the constant cover traffic, it is also hard to analyse the traffic to guess what is going on.

4. Kademlia

Kademlia is properly one of the most widely structured p2p networks in use, why it is an interesting system to be able to attack, and hence to secure.



In each k-buckets, a node only knows *k* of the nodes at a time in that part of the network.

Now whenever we need to route, we XOR the destination with our own key, and the placement of the significant bit tells how many k-buckets away we should look. We will then contact the peers from the given k-bucket, and ask them for the *k* closest nodes it knows, which we then will contact, and so on until we find our destination.

Kademlia prefers to have longliving peers in its k-buckets, and only ever changes them when they become unresponsive. This is a brilliant design choice of the system, where a Sybil attack becomes hard to fulfil. One cannot just spawn an army of peers and join the network to corrupt it, since older peers are preferred.

But still, be do have some weaknesses: the routing can be deterministic, sybils can saturate the network though, and eclipse peers can collude to produce poor routing.

5. S/Kademlia

S/Kademlia seeks to solve this by:

- ***S/Kademlia***
- `Expensive` *They make the system more secure, by making joining more expensive*
- `Signatures on messages` *and lastly, messages are signed*

Making it expensive to generate nodeId, is a way to make S/Kademlia resistant to Sybil and Eclipse attacks. Such expense is empowered by making it difficult to generate many nodes, and impossible to choose node IDs respectively.

This can be done using a centralised certificate authority or by using cryptographic puzzles to generate nodeIds.

At the beginning of the network, when it is small, S/Kademlia proposes using the certificate authority method.

But, lets talk about the cryptographic puzzles, which work by mathematically making nodeId generation difficult. S/Kademlia uses two crypto puzzles, the first is a static one to make it hard to choose node IDs near a target. Lets see why by some pseudo code!

```
while true { // in a loop
    pk, sk = gen() // we generate key pairs
    p = H(H(pk)) // and we double hash the public
    if first c1 bits in p == 0 { // if the first c1 bits == 0
        return pk, sk // we have found our key pair!
    }
}
```

Being that it is the double hash that needs to satisfy some requirement, and our nodeId is a single hash, we make it hard to target an ID.

The dynamic puzzle to make it hard to generate many node IDs, because it needs to find a special number with a property.

```
nodeId = H(pk) // generate our nodeId
while true { // in a loop
    X = random() // make a random number
    p = H(nodeId ⊕ X)
    if first c2 bits in p == 0 {
        return nodeId, X
    }
}
```

And now we have the keys, the nodeid, and the last peace of the puzzle, the number X which proves the node went through a lot of work to find it.

Then, when a peer sends messages, it sends the public key, node ID, and random number X to the current node, and the current node can verify that it used the puzzles in O(1) time.