Anonymous Communication Tools

Govind Kumar
210050058

Khushang Singla
210050085

March - April 2023

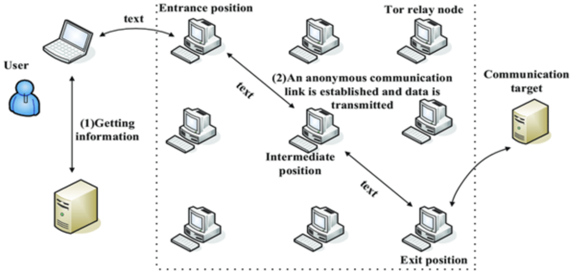
1 Overview

Anonymity describes situations where the acting person's identity is unknown. The important idea here is that a person be non-identifiable, unreachable, or untrackable. Anonymous online services allow people to communicate and share content without revealing their name or true identity. The benefits of these tools include promoting free criticism, security to whistle blowers, prevention of individual data collection, privacy among individuals, etc. Nowadays, there are many tools being built for anonymous communications. Onion routing and Garlic Routing are two of the most famous routing protocols for communicating anonymously on internet. These routing protocols route the data through various nodes so that the path cannot be traced by someone looking the network. VPN (Virtual Private Network) also provide some level of anonymity but in VPNs, user's privacy is in hands of a single person with access to the VPN servers. There also exist remailers and rewebbers to provide anonymous e-mailing service(eg. ProtonMail) and browsing services.

2 Tor - Onion Routing

Tor, short for "The Onion Router," is free and open-source software for enabling anonymous communication. It directs Internet traffic via a free, worldwide, volunteer overlay network that consists of more than seven thousand relays. Although Tor was initially developed by the US government in 2002, it is not presently controlled by any one entity. Tor has been one of the most famous and widely used anonymous communication tool. The release of the Tor Browser made Tor more accessible to everyday internet users and activists. Tor Browser is simply an Internet Browser based on Firefox, with modifications to hide the user's IP address.

2.1 Onion Routing



In a nutshell, onion routing refers to encapsulating message under layers of encryption at different nodes before it reaches the final destination. All the nodes only know about the previous node and the next node. In this way, no single node knows the entire path of the message. Clients choose these path randomly and build a circuit. These circuits change every few minutes preventing any snooping attempts.

2.2 The Tor Design

2.2.1 Onion Router

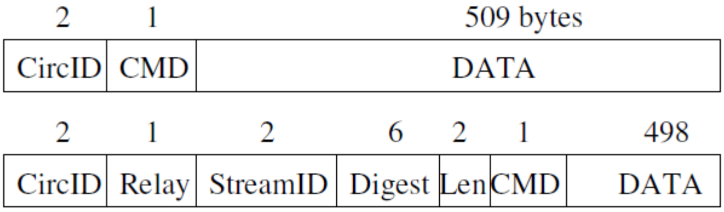
Each onion router runs as a normal user-level process. It maintains a TLS connection to every other onion router in the network. The task of an onion router is to connect to requested destination and relay the data. Each onion router knows only the previous and next onion router in the circuit. Each onion router maintains a long-term identity key and a short-term onion key. The identity key is used to sign TLS certificates and to sign the router descriptor. Router descriptor is a summary of its keys, address, bandwidth, exit policy, and so on. Directory servers use identity key to sign directories. The onion key is used to decrypt requests from users to set up a circuit and negotiate ephemeral keys.

2.2.2 Onion Proxy

Each user runs a Tor client called an Onion Proxy on their computer. The Onion Proxy is responsible to fetch directories, building circuits, and sending messages through the circuits. These onion proxies accept TCP streams and multiplex them across the circuits.

2.2.3 Cells

Onion routers communicate with one another, and with users Onion Proxies, via TLS connections with ephemeral keys. Traffic passes along these connections in fixed-size cells. Each cell is 512 bytes, and consists of a header and a payload.



2	1	509 bytes
CircID	CMD	DATA

2	1	2	6	2	1	498
CircID	Relay	StreamID	Digest	Len	CMD	DATA

CircID is different for each hop (OP/OR or OR/OR) in the circuit.

CMD specifies the type and action of the cell.

CONTROL cells are used to manage circuits and connections. These are interpretted directly by the node receiving it. Some of the control cells command are create, created, destroy, padding, etc.

RELAY cells carry end-to-end stream data. They have additional header fields to specify the stream ID, a checksum to maintain integrity, length of the relay payload and a relay command. Some of relay cell commands are relay begin, relay data, relay end, relay teardown, etc.

2.2.4 Circuits and streams

A circuit is a path through the Tor network. Each circuit can be shared by many TCP streams. These circuits are built periodically by client's OP and are torn down after their expiry. Each stream is a sequence of relay cells traveling along a circuit.

2.2.5 Directory Servers

Tor uses a small group of redundant, well-known onion routers to track changes in network topology and node state, including keys and exit policies. Each such directory server acts as an HTTP server, so clients can fetch current network state and router lists, and so other ORs can upload state information. Onion routers periodically publish signed statements of their state to each directory server. The directory servers combine this information with their own views of network liveness, and generate a signed description (a directory) of the entire network state. Client software is pre-loaded with a list of the directory servers and their keys, to bootstrap each client's view of the network.

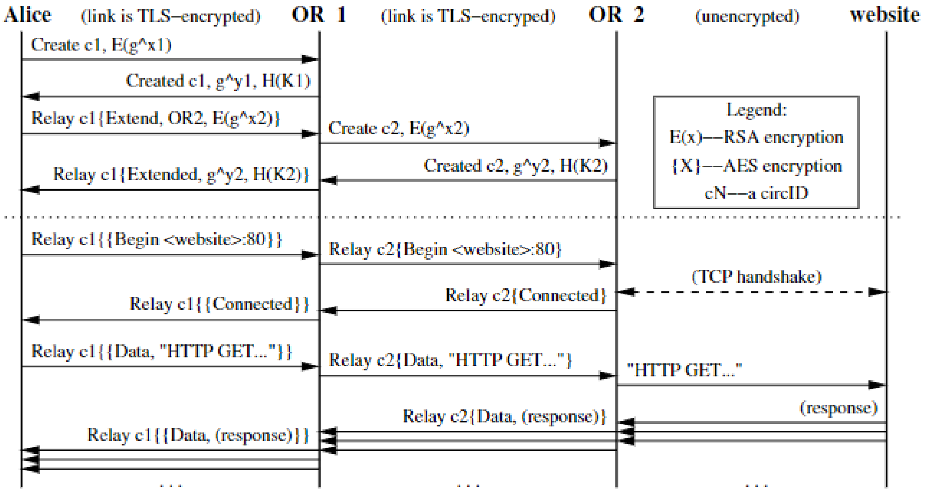
2.2.6 Rendezvous Points and hidden services

Rendezvous points are a building block for location-hidden services. Location-hidden services allow a person to offer a TCP service, such as a webserver, without revealing his IP address.

2.3 Constructing a Circuit

A user's OP constructs circuits incrementally, negotiating a symmetric key with each OR on the circuit, one hop at a time. To create a circuit, the OP (call her Alice) sends a CREATE cell to an OR of her choice (call him Bob) on the circuit containing the first half of a Diffie-Hellman handshake encrypted using OR's public key. The OR replies with a CREATED cell containing the second half of the handshake and a hash of the negotiated key. Further comunication between them is encrypted using the negotiated key.

To extend the circuit further, ALice sends a relay EXTEND cell to Bob, containing the address of the next OR and the first half of a new Diffie-Hellman handshake encrypted using that OR's public key. Bob copies the first half of the handshake into a CREATE cell and sends it to the next OR. When the next OR replies with a CREATED cell, Bob wraps into relay extended cell and pass it back to Alice.



This circuit-level handshake protocol achieves unilateral entity authentication.

2.4 Role of Relay Cells

Upon re-ceiving a relay cell, an OR looks up the corresponding circuit, and decrypts the relay header and payload with the session key for that circuit. If the cell is headed away from Alice the OR then checks whether the decrypted cell has a valid digest. If valid, it accepts the relay cell and processes it as described below. Otherwise, the OR looks up the circID and OR for the next step in the circuit, replaces the circID as appropriate, and sends the decrypted relay cell to the next OR.

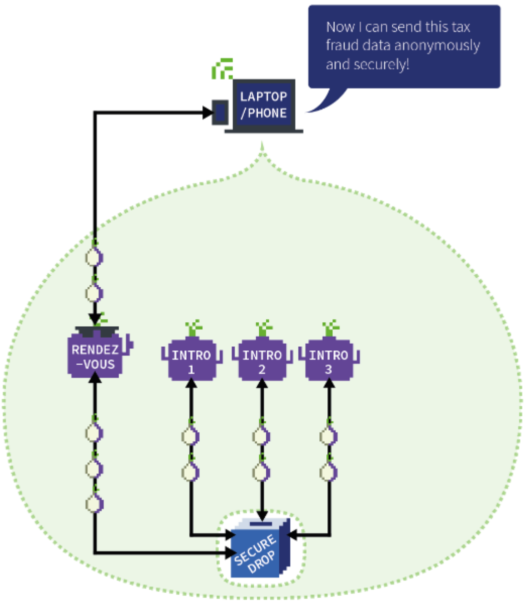
OPs treat incoming relay cells similarly: they iteratively unwrap the relay header and payload with the session keys shared with each OR on the circuit, from the closest to farthest. If at any stage the digest is valid, the cell must have originated at the OR whose encryption has just been removed.

To construct a relay cell addressed to a given OR, Alice assigns the digest, and then iteratively encrypts the cell payload (that is, the relay header and payload) with the symmetric key of each hop up to that OR.

When an OR later replies to Alice with a relay cell, it encrypts the cell's relay header and payload with the single key it shares with Alice, and sends the cell back toward Alice along the circuit. Subsequent ORs add further layers of en- cryption as they relay the cell back to Alice.

To tear down a circuit, Alice sends a destroy control cell. Each OR in the circuit receives the destroy cell, closes all streams on that circuit, and passes a new destroy cell forward. These circuits can also be torn down incremently using relay truncate cell.

2.5 Rendezvous points in Tor


Now I can send this tax fraud data anonymously and securely!
LAPTOP /PHONE
RENDEZ -VOUS
INTRO 1
INTRO 2
INTRO 3
SECURE DROP

The following steps are performed on behalf of Alice and Bob by their local OPs to communicate using a rendezvous point:

- Bob generates a long-term public key pair to identify his service.
- Bob chooses some introduction points, and advertises them on the lookup service, signing the advertisement with his public key. He can add more later.

- Bob builds a circuit to each of his introduction points, and tells them to wait for requests.
- Alice learns about Bob's onion service out of band. She retrieves the details of Bob's service from the lookup service. If Alice wants to access Bob's service anonymously, she must connect to the lookup service via Tor.
- Alice chooses an OR as the rendezvous point (RP) for her connection to Bob's service. She builds a circuit to the RP, and gives it a randomly chosen "rendezvous cookie" to recognize Bob.
- Alice opens an anonymous stream to one of Bob's intro- duction points, and gives it a message (encrypted with Bob's public key) telling it about herself, her RP and ren- dezvous cookie, and the start of a DH handshake. The introduction point sends the message to Bob.
- If Bob wants to talk to Alice, he builds a circuit to Al- ice's RP and sends the rendezvous cookie, the second half of the DH handshake, and a hash of the session key they now share.
- The RP connects Alice's circuit to Bob's. Note that RP can't recognize Alice, Bob, or the data they transmit.
- Alice sends a relay begin cell along the circuit. It arrives at Bob's OP, which connects to Bob's webserver.
- An anonymous stream has been established, and Alice and Bob communicate as normal.

3 Invisible Internet Project - I2P

This is a project trying to implement **Garlic Routing** - A Routing protocol built over onion routing. It is not used widely as of today because it needs slightly more technical knowledge to set up for the first time for use. Also, the sites outside the network of invisible internet cannot be accessed through the invisible internet. One may access outside internet through proxies (which do exist) but the proxies may be malicious and it may not be safe to do so. This also is a big problem hindering the popularity of i2p.

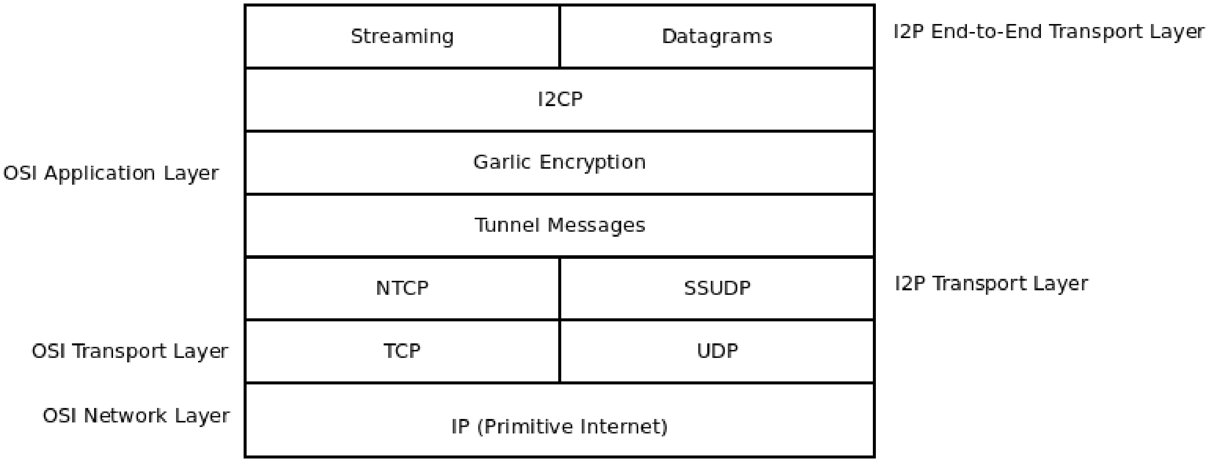
3.1 I2P Protocol

I2P adds more layers in Application Layer of primitive internet to introduce anonymity. Figure 1 shows the layers in this stack.

I2P transport layer is built over Transport layer of regular internet. It is strictly for next-hop transfer among I2P routers. This is non-anonymous transfer. This layer has two protocols built over TCP and UDP each.

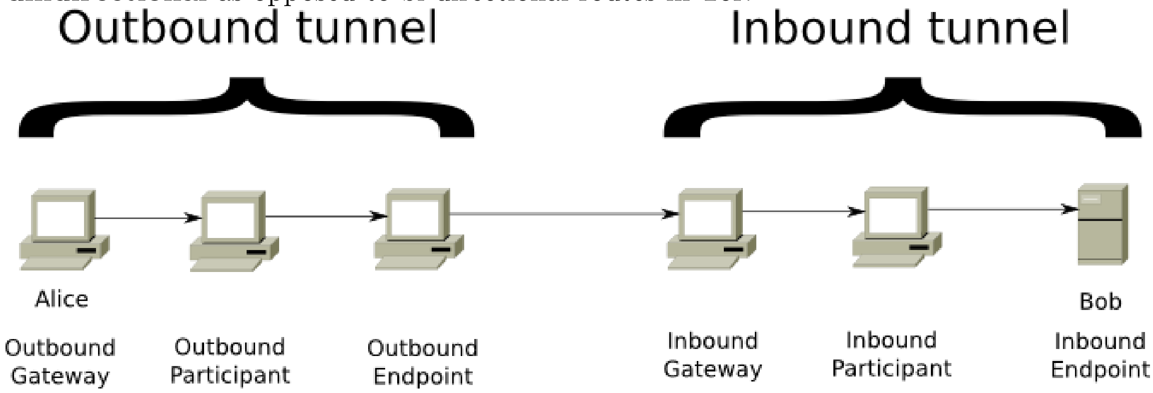
- NTCP2 - New I/O based TCP
- SSU - Secure and Semi-Reliable UDP

Figure 1: I2P Protocol Stack



3.2 Structure of I2P: Components

- **Tunnel**
  Messages are sent from one node to another through tunnels. There are two tunnels, outbound tunnel and inbound tunnel. This is needed as per Garlic Routing protocol. The sender first builds an outbound tunnel. It gets the details of Inbound tunnel from netDB. In a tunnel a gateway refers to first router and the last router is called endpoint. A user might have multiple such outbound and inbound tunnels. These tunnels used in I2P are **unidirectional** as opposed to bi-directional routes in Tor.



  The sender adds routing instructions with the message, encrypts it and sends it through the tunnel. Just like Onion routing, this message is also encrypted using layered encryption. When the endpoint recieves the final message, it gets the routing instruction to the inbound gateway of reciever. The inbound gateway then sends this message to inbound endpoint through the tunnel. Except for this difference, if we see tunnels as gateway to endpoint, both work in a similar way.

  Gateway accumulates some messages to be sent through the tunnel, adds path to reciever and converts them to **Garlic Message** so that they can be sent through tunnel. When the endpoint of tunnel finally decrypts the message, it separates the messages and forwards them to the required hosts.

- **Network DataBase**
  Network Database stores the information about the routers present in the network. It also stores information about the tunnel gateways for inbound tunnels of users. In I2P, routers

are identified by their public keys. This Network Database is a decentralized database. When a user wants to communicate with some other router in the Invisible Internet, he needs to lookup in the network database to find the details of inbound tunnel gateway for reciever. As it is a decentralized database, possibly, a router may not have access to complete database at any given time. As I2P is not widely used, it is possible to find these details in a few tries but if the usage increases, it might become difficult to access this information.

Network database stores two kinds of data: **leaseSets**(Section 3.3.2) and **routerInfo**(Section 3.3.3)

3.3 Network Database

3.3.1 Distributed Storage

Network Database (NetDB) is a distributed storage. The storage is distributed among I2P nodes. A special set of nodes called **floodfill** nodes are used for storing the data. There is no fixed set of floodfill nodes. The users of I2P can Opt-in as Floodfill routers. Any router which wants to publish it's routerInfo does so by sending the data to the nearest floodfill router. The information of whether a router is a floodfill router is stored in routerInfo of the corresponding router. When a floodfill router gets a DatabaseStoreMessage, it floods it.

For flooding, it looks up several floodfill routers closest to the routing key of NetDb entry, where routing key is SHA256 hash of routerIdentity/Destination with date appended.

Unlike Tor, these floodfill peers need not be trusted, and may change over time. Usually, the floodfill routers are one with large bandwidth availability. The only extra work they need to do is **respond to netDB stores and queries**.

3.3.2 Lease Sets

Lease Sets are used to document tunnel entry points for a particular client destination. The following information is stored in a Lease Set.

Lease stores information of the inbound tunnel gateway. A lease stores the following information. It is needed to send messages to the Destination.

Gateway Router is specified for the tunnel. It is specified by spcifying it's identity.

Tunnel ID to be used to send message through the tunnel.

Expiry Date stores the time till when the tunnel is available.

Destination encryption key, signing key and a certificate.

Additional encryption public key for use in encrypting garlic messages. It is used for end-to-end ElGamal/AES + Session Tag encryption.

Signature of the lease set to ensure that the data is published by the entity mentioned in destination.

There are various types of Lease sets like Unpublished Lease sets, Encrypted LeaseSets etc.

3.3.3 Router Info

RouterInfo includes the following details

Router's Identity stores an encryption key, a signing key and a certificate to authorize that. The public_key is used for ElGamal Encryption in next-hop messages. The signing public key and key certificate are used for verifying signatures.

Contact Addresses where the router can be reached. It contains mapping of transport protocol (NTCP or SSU) with ip address and port.

Publish Date is the time when this info was published.

Options is a set of arbitrary options for telling the bandwidth capacities, router version, netId. There are some other stat options also.

Signature of all the above data.

Router Info is stored in NetDB. This is required for building tunnels.

3.4 Tunnels

Tunnels can be divided into two types of tunnels, namely inbound tunnels and outbound tunnels. The creator of the tunnel is the only one who knows all participants of tunnels. There are three types of routers in tunnels, namely gateway, participants, and endpoint. The gateway collects all messages, combines them and sends to next hop. The participant routers just decrypt/encrypt and forward to nexthop. They don't even know if it is an inbound or outbound tunnel. The endpoint recieves the messages, opens it and forwards it, either to another router, or to a tunnel gateway or locally.

Building a Tunnel

For building a tunnel, we need to ensure that the hops can't be associated to each other. Only the next hop peers should be able to know that they are in same tunnel. Every hop in the tunnel gets a random non-zero tunnel ID. Every record gets a random tunnel IV key, reply IV, layer key and reply key. The record contains the tunnel ID, the next hop tunnel ID, routerid hash, tunnel layer key and iv key, reply key and iv and some uninterpreted padding to send to next hop. The tunnel creation is accomplished by a single message passed along the path of peers in the tunnel, rewritten in place, and transmitted back to the tunnel creator. This is made up of variable number of records, each record potentially for each peer along the path. For building an inbound tunnel, an outbound tunnel is used to send this message whereas it is directly sent as next hop for building outbound tunnel. Every hop replaces it's record by it's reply and the endpoint is instructed to send the message to the tunnel creator.

A gateway first does some message pre-processing and then sends it the next hop after gateway encryption. In this pre-processing, it takes the collected I2NP messages, makes {delivery instructions,message} pairs, adds tunnel ID, IV, checksum and padding. It then picks an IV, iteratively encrypts it and the message as needed and forwards that to next hop with tunnel ID and IV.

When a participant recieves a tunnel message, it reads the tunnel id, maps it with next hop tunnel id and sends it to the next hop. The participant encrypts the recieved IV with AES256ECB using their IV key to determine the current IV and uses that IV with it's layer key to encrypt that data. It then encrypts the current IV with AES256/ECB using their IV key again, then forwards the tuple {nextTunnelId,nextIV,encryptedData} to the next hop.

For an outbound tunnel, the gateway just encrypts data using it's layer key to get the preprocessed data. Whereas for inbound tunnels, the endpoint needs to decrypt that according to each hop's layer key and IV to get the pre-processed data.

3.5 Sending Messages

For sending a message, a router needs an outbound tunnel. For this, a tunnel is created. For creating a tunnel, the router requests routerInfo from netDB through Database. It selects routers to use in the tunnel and builds a tunnel. Then it uses this outbound tunnel to request leaseSet of destination from Database. This leaseSet is used to get the inbound tunnel gateway routerIdentity. It then puts delivery instruction for the message as this gateway router's information and tunnel id. This is then encrypted accordingly and sent to next hop. When it reaches the endpoint of this tunnel, it gets the forwarding instructions. The message is forwarded to the gateway which then sends it to the required destination. For a reply, the sender needs to put it's lease or Destination information in the message sent.

4 VPN

VPN stands for Virtual Private Network. VPNs provide a set of servers through which the connected clients route their traffic. The cost to maintain VPN services is high as with increase in number of clients, the service provider needs to get more servers to keep up with the load. This reduces scalability of VPNs.

One bigger problem of VPN is that the admin of the VPN servers can see what each user is accessing, thus VPNs can't be considered enough anonymous. VPNs are mostly used to be able to access a private network from anywhere and less popular to be used for privacy or anonymity over internet. But they may well be used to hide from the websites, who is trying to access the content as for the websites see the VPN servers as their clients.

5 Resources Used

1. An Analysis of Tools for Online Anonymity

2. Tor Website

3. Tor: The Second-Generation Onion Router

4. I2P Docs