Project Report - Milestone 2 for Computer Security And Privacy

Aakanksha Agrawal

April 28, 2023

1 Project Goals

The project aims to achieve the following goals:

1. A complete understanding of Tor and TCP-based anonymisation (based on paper - Tor: The Second-Generation Onion Router).
2. Acquire and run the actual Tor protocol, gain the ability to run a few applications over Tor on both iOS and Linux.
3. Host an Onion Service
4. Show packet capture of the Tor protocol through wire-shark
5. Write networking code that implements some central parts of the Tor protocol (in Rust or Python) and demonstrate TCP anonymisation for web browsing.
6. Understand and present:
   (a) Network security analysis of the protocol.
   (b) Cryptographic security analysis of the protocol.
7. (If time permits) Submit a patch to the actual codebase.

2 Introduction

2.1 Motivation and backgroud

Goals:

1. Alice and Bob want to establish communication with information security ie. Data integrity, data authentication, end-point authentication.

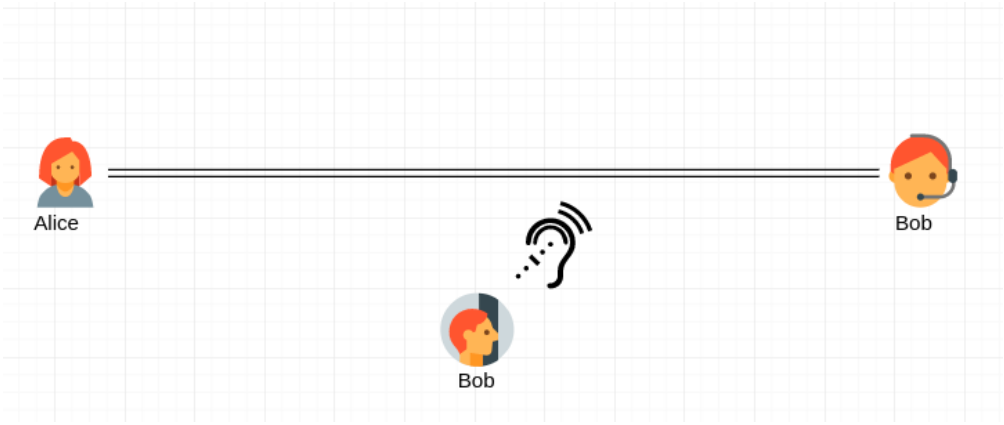


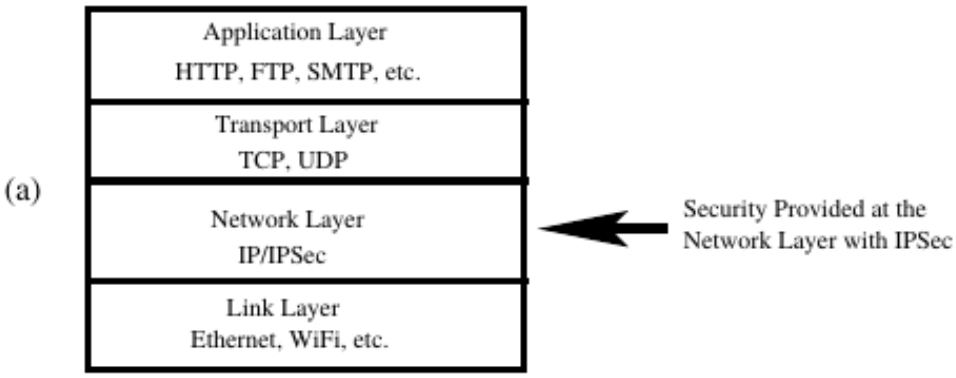
Figure 1: Alice and Bob talking on a secure channel. Eve, a passive adversary snooping.

2. Eve or any passive adversary snooping on the enroute packet traffic should not be able to analyze the packet headers for the purpose of finding out who was talking to whom. Gleaning information regarding the original source of the packets and their ultimate destination is referred to as **traffic analysis attack**

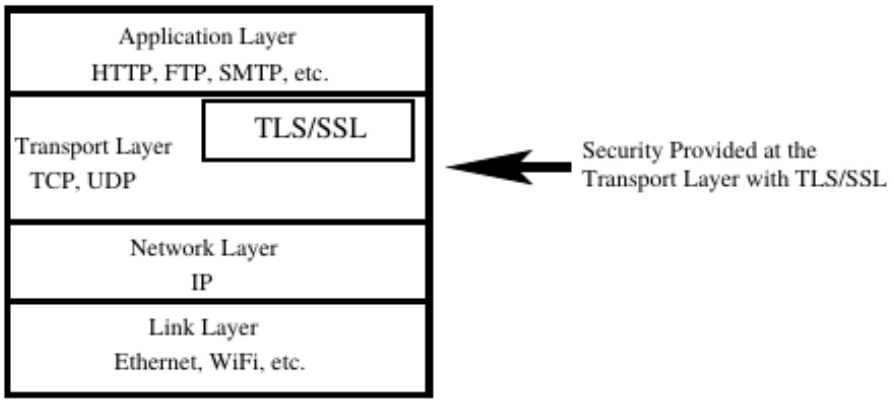
Real life use case? Alice, a reporter living in an authoritarian state, wants to share her report with a news organization like the New York Times, but she does not want anyone other than the New York Times to know. If the state discovers that Alice is communicating with the New York Times, she may face consequences. To ensure her anonymity, Alice needs to find a solution.

2.2 Other approaches and why they do not work

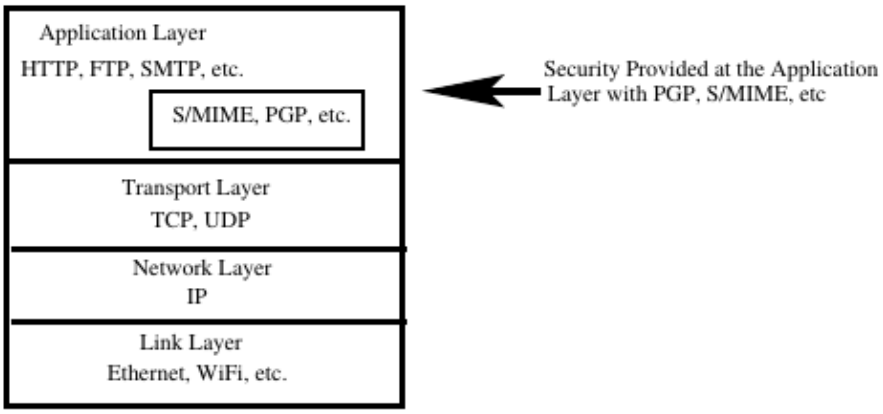
Confidentiality and authentication for information security can be provided in three different layers in the TCP/IP protocol stack, as shown in this figure [5].



(b)



(c)



- Information security provided at the **application layer**, such as through the use of PGP, still transmits source and IP addresses in clear text. Thus linking Alice to Bob
- when protocols based on TLS are used for establishing encrypted communication channels for the transfer of information between the web browsers and the web servers, the IP packet headers are always in clear text.Thus, linking Alice to Bob
- (in progress, I'm still trying to contrast IP Sec and Tor for protecting against traffic analysis attacks)

  VPNs can help you remain anonymous but it depends on situation to situation. Let's say Alice is in Country A, and want's to connect to connect to a webserver in Country B. Let's say Country A's firewall is blocking all traffic with destination IP addresses in Country B. Alice will have to first connect to her VPN server which has to be located outside of Country A or Country B, so eventually connect to the desired webserver in Country B.
- VPN adoption has seen steady growth over the past decade due to increased public awareness of privacy and surveillance threats. In response, certain governments are attempting to restrict VPN access by identifying connections using "dual use" Deep

Packet Inspection technology. [10] . Binxing Fang, the designer of the Great Firewall of China (GFW) said there is an "eternal war" between the Firewall and VPNs, and the country has ordered ISPs to report and block personal VPN usage.

2.3 Presenting: Tor-the next gen onion router

- The Tor protocol for anonymized routing is described in the paper "Tor: The Second-Generation Onion Router" by Roger Dingledine, Nick Mathewson, and Paul Syverson that was presented at the 13th Usenix Security Symposium in 2004. [4]
- Tor is open-source and available to all from http://www.torproject.org [1]
- Although originally an acronym standing for "The Onion Router," "Tor" is now used as a name unto itself.

2.4 Mix networks and David Chaum

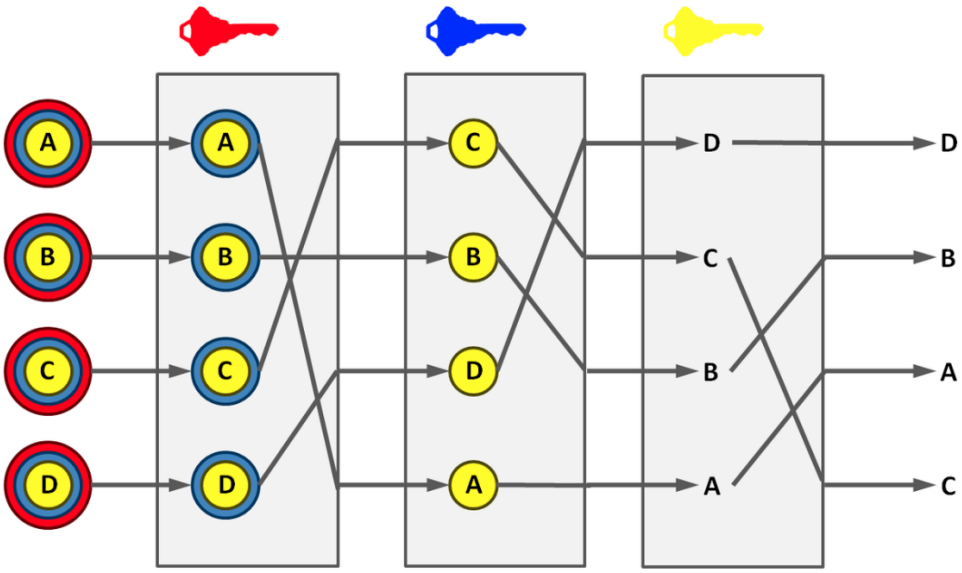
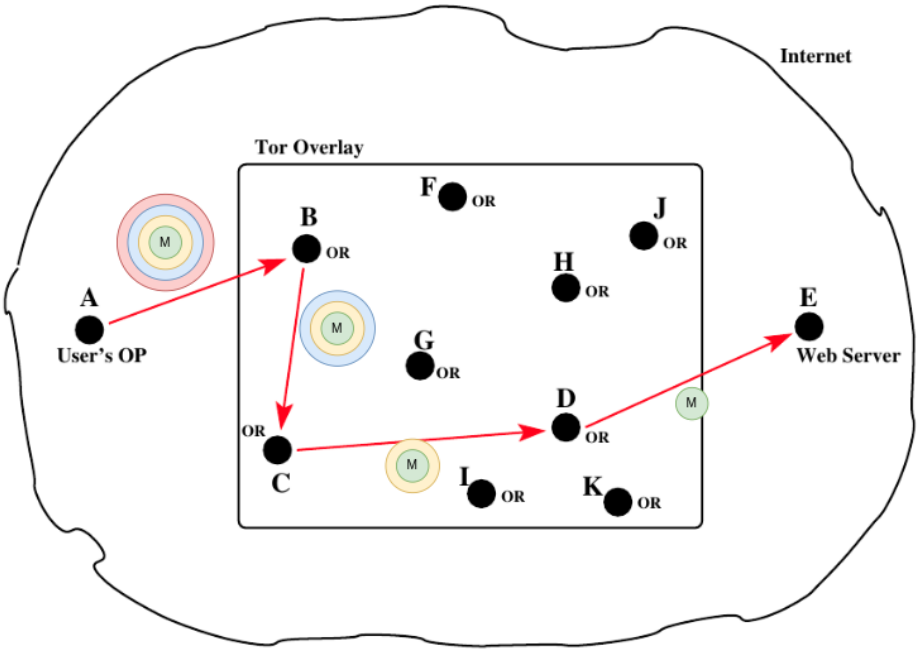


Figure 2: A simple decryption matrix
[9]

- David Chaum published the concept of Mix Networks in 1979 in his paper: "Untraceable electronic mail, return addresses, and digital pseudonyms" [3].
- Figure 2 is a Simple decryption mix net. Messages are encrypted under a sequence of public keys. Each mix node removes a layer of encryption using its own private key. The node shuffles the message order, and transmits the result to the next node. [9]
- Each node kind of peels a layer of an onion to eventually reach the onion bulb. This is the nomenclature for "Onion" routing comes from.

2.5 Big Picture- TOR



- - The Tor protocol is based on the twin notions of Client nodes (Onion Proxy/OR) and Relay nodes (called Onion Router/OR) . Alice, our User node first queries a Tor directory for the IP addresses of the Relay nodes in the Tor overlay.
- The User Node then selects a subset of these Relay nodes, usually just 3, for constructing a path to the destination resource. This path constitutes a circuit.
- Subsequently, the two parties at the two end of a circuit may use it for an arbitrary number of TCP streams.
- In this each node in the path knows its predecessor and successor nodes, but no other nodes in the circuit i.e. each node on the path has only local knowledge concerning the overall path
- Traffic flowing down the circuit is sent in fixed-size 'cells'(usually 512 bits), which are unwrapped by a symmetric key at each node (like the layers of an onion) and relayed downstream.

3 The basic Tor Design:

Tor is a distributed overlay network designed to anonymize low-latency TCP-based applications such as web browsing, secure shell, and instant messaging: Tor network is an overlay network consisting of

1. Relay nodes(called Onion Router): These are nodes of the network which are used to hop around.

    - Each Relay node runs a **normal user-level process without any special privileges**. Let's call this user-level process Onion Router
    - Each Onion Router maintains a TLS connection with every other Onion Router
    - Each Onion router maintains a Long Term identity key used to Sign TLS certificates, Sign OR's router descriptor, sign Sign directories. Short Term Onion key used to Decrypt requests from users, Setup a circuit, Negotiate ephemeral keys. A short-term "Connection key" used to negotiate TLS connections

2. Client Nodes(also called Onion Proxy in the literature): Each user/client runs a local software called an Onion Proxy to fetch directories, establish circuits across the network, and handle connections from user applications.

    - These Onion Proxies accept TCP streams(via socks) from other user applications and multiplex them across the circuit

3. **directory authority:**

    - It is a special-purpose relay that maintains a list of currently-running relays and periodically publishes a consensus together with the other directory authorities. [5]
    - Each Tor non-exit and exit relay sends information about itself to these Directory Authority servers once every 18 hours. The Directory Authority servers compile this information and publish a list of all the current non-exit and exit relays once every hour. [5]

‘

3.1 Threat Model

Tor does not protect against a global passive adversary. Instead, it assumes an adversary who can observe some fraction of the network traffic, generate, modify, delete, or delay traffic, operate onion routers of his own, and compromise some fraction of the onion routers.

3.2 Cells

Traffic passes in fixed-length (512 bytes) cells and consists of a header and a payload. The header includes a circuit identifier(circID) that specifies which circuit the cell refers to, and a command that specifies what to do with the cell's payload. Based on the command field, there are 2 kinds of cells:

1. Control cells: used to manage circuits and connections

    The second field, CMD, in a control cell is a one-byte integer representation of a command. A control cell may contain the following different commands:

```
/** List of default directory authorities */

static const char *default_authorities[] = {
  "moria1 orport=9101 "
    "v3ident=D586D18309DED4CD6D57C18FDB97EFA96D330566 "
    "128.31.0.39:9131 9695 DFC3 5FFE B861 329B 9F1A B04C 4639 7020 CE31",
  "tor26 orport=443 "
    "v3ident=14C131DFC5C6F93646BE72FA1401C02A8DF2E8B4 "
    "ipv6=[2001:858:2:2:aabb:0:563b:1526]:443 "
    "86.59.21.38:80 847B 1F85 0344 D787 6491 A548 92F9 0493 4E4E B85D",
  "dizum orport=443 "
    "v3ident=E8A9C45EDE6D711294FADF8E7951F4DE6CA56B58 "
    "194.109.206.212:80 7EA6 EAD6 FD83 083C 538F 4403 8BBF A077 587D D755",
  "Bifroest orport=443 bridge "
    "37.218.247.217:80 1D8F 3A91 C37C 5D1C 4C19 B1AD 1D0C FBE8 BF72 D8E1",
  "gabelmoo orport=443 "
    "v3ident=ED03BB616EB2F60BEC80151114BB25CEF515B226 "
    "ipv6=[2001:638:a000:4140::ffff:189]:443 "
    "131.188.40.189:80 F204 4413 DAC2 E02E 3D6B CF47 35A1 9BCA 1DE9 7281",
  "dannenberg orport=443 "
    "v3ident=0232AF901C31A04EE9848595AF9BB7620D4C5B2E "
    "193.23.244.244:80 7BE6 83E6 5D48 1413 21C5 ED92 F075 C553 64AC 7123",
  "maatuska orport=80 "
    "v3ident=49015F787433103580E3B66A1707A00E60F2D15B "
    "ipv6=[2001:67c:289c::9]:80 "
    "171.25.193.9:443 BD6A 8292 55CB 08E6 6FBE 7D37 4836 3586 E46B 3810",
  "Faravahar orport=443 "
    "v3ident=EFCBE720AB3A82B99F9E953CD5BF50F7EEFC7B97 "
    "154.35.175.225:80 CF6D 0AAF B385 BE71 B8E1 11FC 5CFF 4B47 9237 33BC",
  "longclaw orport=443 "
    "v3ident=23D15D965BC35114467363C165C4F724B64B4F66 "
    "ipv6=[2620:13:4000:8000:60:f3ff:fea1:7cff]:443 "
    "199.254.238.52:80 74A9 1064 6BCE EFBC D2E8 74FC 1DC9 9743 0F96 8145",
  NULL
};
```

Figure 3: Caption

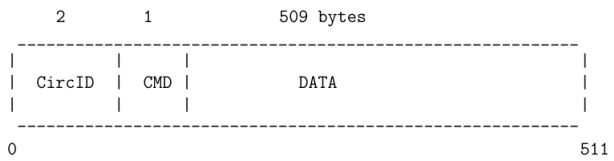


Figure 4: Control Cell

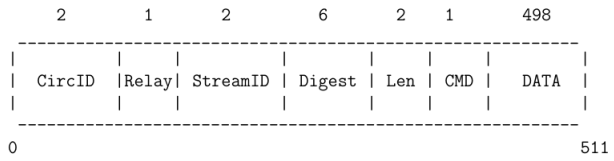


Figure 5: Relay Cell

create : sent by an OP or OR to another OR to extend the path to the next node
created : when an OR successfully extends the path to the next node in response to a create command from the previous node on a path, it sends back a created message to the previous node.
destroy : sent by a node to another node to teardown the path
padding : used for "keepalive" when a timeout might shut down a circuit otherwise

2. Relay cells: used to transfer user data between Onion Routers Relay cells have an additional header containing: **stream ID:** it's a stream identifier, as many streams can be multiplexed over a single circuit
Digest: and end to end checksum for integrity checking
Length of the payload
Relay command: The 1-byte command field (CMD) in the header of a relay cell can be used to create following kinds of such packets:
relay extend : to extend the circuit by one hop
relay extended : to notify that relay extend was successful
relay truncate : to drop the last the OR on the path
relay truncated : to notify that relay truncate was successful
relay begin : to open a new stream
relay connected : to notify the OP that a stream was successfully opened
relay end : to close a previously opened stream
relay data : for transmission of data in stream
relay sendme : used for congestion control
relay teardown : used to close a broken stream

The entire content of the relay header and the relay payload are encrypted an decrypted together as the relay cell moves along the circuit using 128-bit AES cipher in counter

mode to generate a stream.

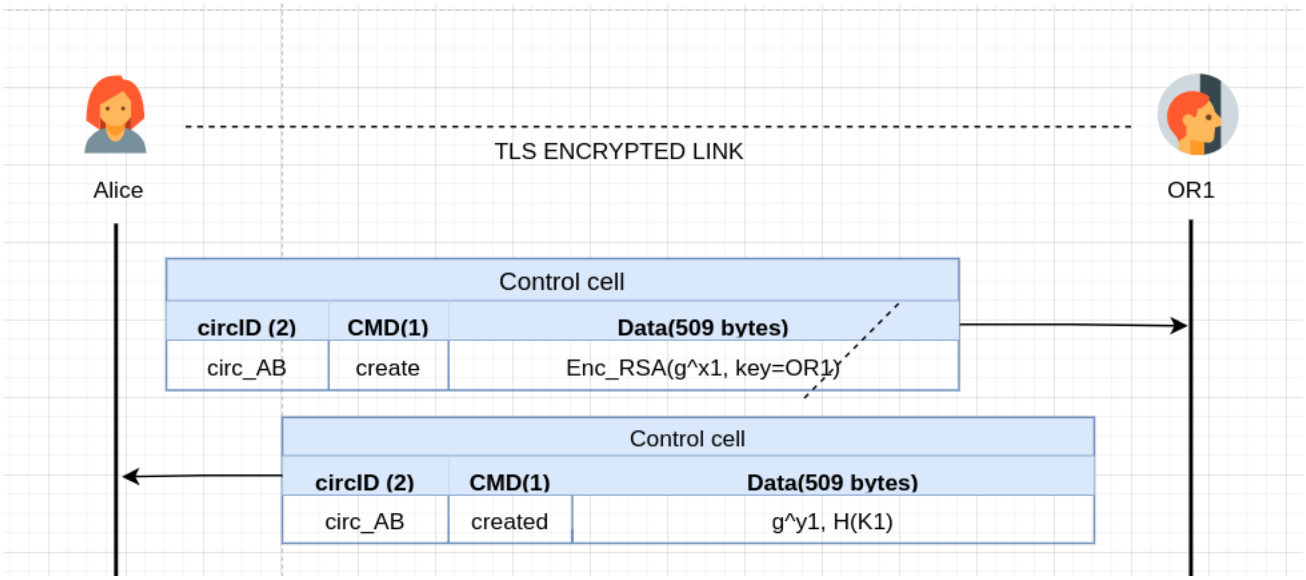
3.3 Querying a DA and choosing relay nodes

- First, a user's Onion Proxy queries a special node called the directory server to get a list of all public relay nodes(ORs). Out these the user chooses a minimum of 3 relay nodes to make a circuit.

- Anyone host can become a relay node by `running tor in relay mode`. An interactive map of relay nodes in the world : https://torflow.uncharted.software/ showing relay nodes and traffic:. When you run `tor in relay mode`, you run a process(Onion Router) on a port of your choice. Your public ID, this port number and an email ID among other things that you publish to the worls(so, the list of all relay nodes is public). Currently, there are about 6000 relays in the world [2]. You can query the latest list of relays with

```
curl -s 'https://onionoo.torproject.org/details?type=relay' | jq . | less
```

3.4 Constructing a Circuit:



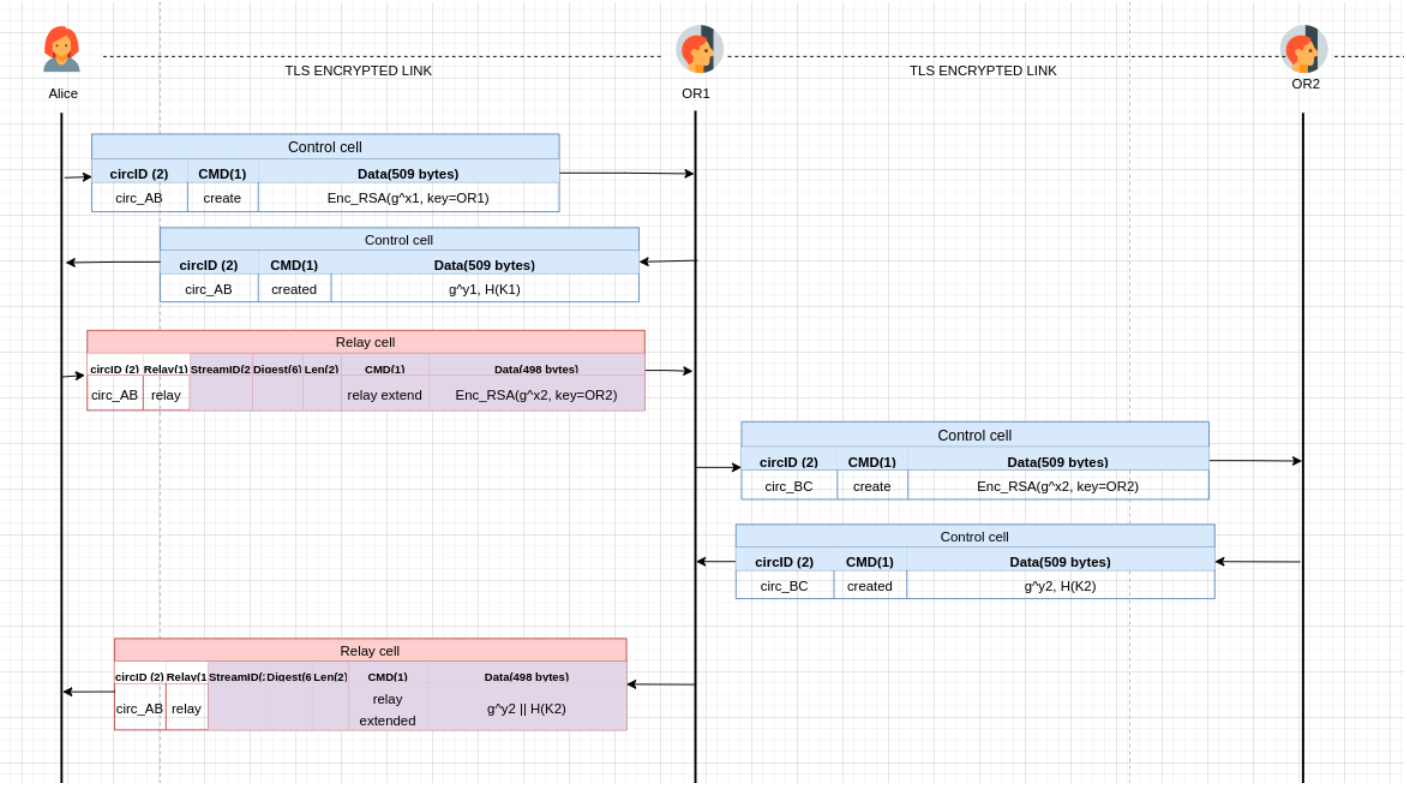
A user's Onion Proxy constructs circuits incrementally, negotiating a symmetric key with each Onion Router one hop at a time.

- To begin creating a new circuit, the Onion Proxy (call her Alice) sends a create cell to the first node in her chosen path (call him Bob). (She chooses a new circID C_AB not currently used on the connection from her to Bob.) The create cell's payload contains the first half of the Diffie-Hellman handshake (g^x), encrypted to the onion key of the Onion Router (call him Bob).

- Bob responds with a created cell containing g^y along with a hash of the negotiated key $K = g^{xy}$.

Once the circuit has been established, Alice and Bob can send one another relay cells encrypted with the negotiated key.

3.5 Extending the Circuit:



- To extend the circuit further, Alice sends a relay extend cell to Bob, specifying the address of the next Onion Router (call her Carol) and an encrypted g^{x_2} for her.
- Bob copies the half-handshake into a create cell and passes it to Carol to extend the circuit. Bob chooses a new circID C_{BC} not currently used on the connection between him and Carol. Alice never needs to know this circID; only Bob associates C_{AB} on his connection with Alice to C_{BC} on his connection with Carol. This fact plays an important role in ensuring that each node on the path has only the local knowledge of the path
- Carol responds with a created cell to Bob. Bob wraps the payload into a relay extended cell and passes it back to Alice.

Now, the circuit is extended to Carol, and Alice and Carol share a common key $K2 = g^{x_2 y_2}$. An end to end circuit is now constructed and we can begin exchanging data

3.6 (work in progress)Cryptographic analysis(discussed informally):

Assume Bob is the OR1

1.$(G, q, g) \leftarrow \texttt{GroupGen}(1^n)$
2.$\alpha \leftarrow \{2, ...q-1\}$
3.$h_1 \leftarrow g^\alpha$
4.$enc\_h_1 = \pi.enc(g^\alpha, pk_B^e)$

Give $(G, q, g), enc\_h_1$ to Bob

5.$h_1' = \pi.dec(enc\_h_1, sk_B^e)$
6.$\beta \leftarrow \{2, ..., q-1\}$
7.$h_2 \leftarrow g^\beta$
8.$key_2 = h_1'^\beta = g^{\alpha\beta}$
9.$comp : hash_2 = H(key_2)$

Give $h_2, hash$ to Alice

comp: 10.$key_1 = H_2^\alpha = g^{\alpha\beta}$
comp: 11.$comp : hash_1 = H(key_1)$
12.$hash_1 == hash_2$: circuit created

Unilateral entity authentication: this protocol achieves unilateral entity authentication as Alice know's she is handshaking with an OR but the OR does not know anything about Alice as **Alice uses no public RSA key** and remains anonymous). The user Alice remains anonymous to all the ORs in the circuit. But all the ORs in a circuit are known to the user Alice (not surprising, since A chose them for the circuit).

Unilateral key authentication Alice and OR agree on a key, and Alice knows only the OR learns it. In step 4 she encrypts the part of the key to OR's public key and in step 5, OR proves to her that it was indeed him who recieved g^α and him who chose g^β

Forward security and key freshness:

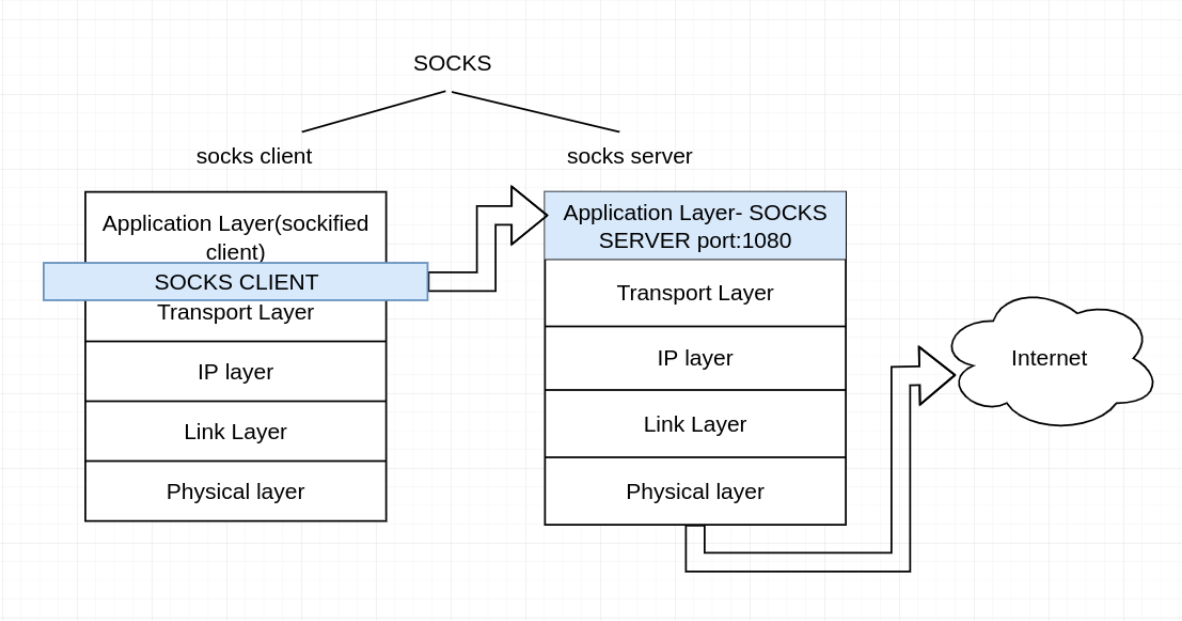
3.7 Before moving ahead- SOCKS

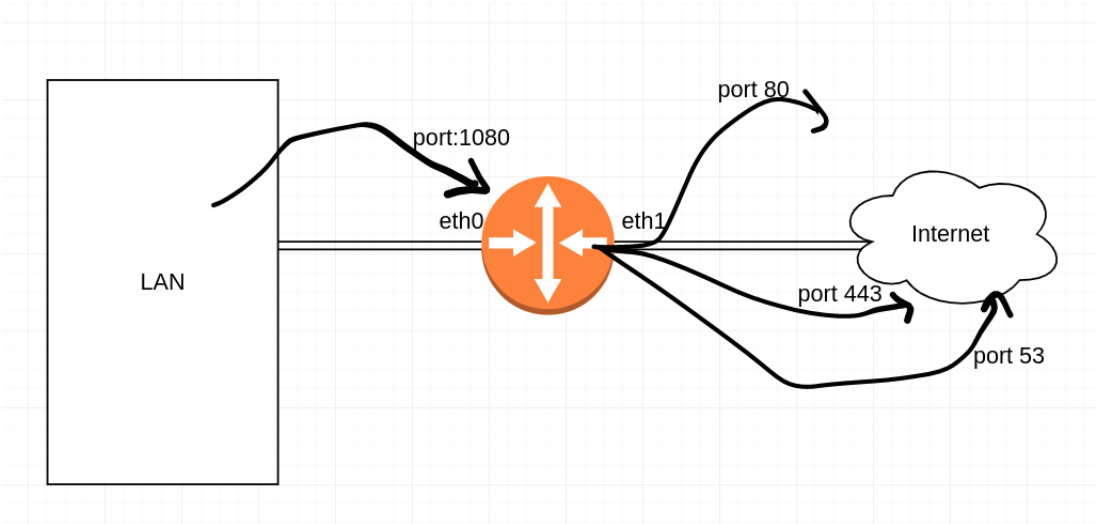
Note: Most of this section is cited from Lecture Notes by Prof Avinash Kak [5]. They were instrumental in understanding this.

After a path is constructed, the user's Onion Proxy(running at the Application Layer), accepts TCP streams from other user applications via socks, and multiplexes them across the circuits. What is socks?
SOCKS is referred to as a generic proxy protocol for TCP/IP based network applications.

- SOCKS, an abbreviation of "SOCKetS", consists of two components: A SOCKS client and a SOCKS server.
- It is the socks client that is implemented between the application layer and the transport layer; the socks server is implemented at the application layer.

- The socks client wraps all the network-related system calls made by a host with its own socket calls so that the host's network calls get sent to the socks server at a designated port(usually 1080), usually 1080. This step is usually referred to as **socksifying the client call**.



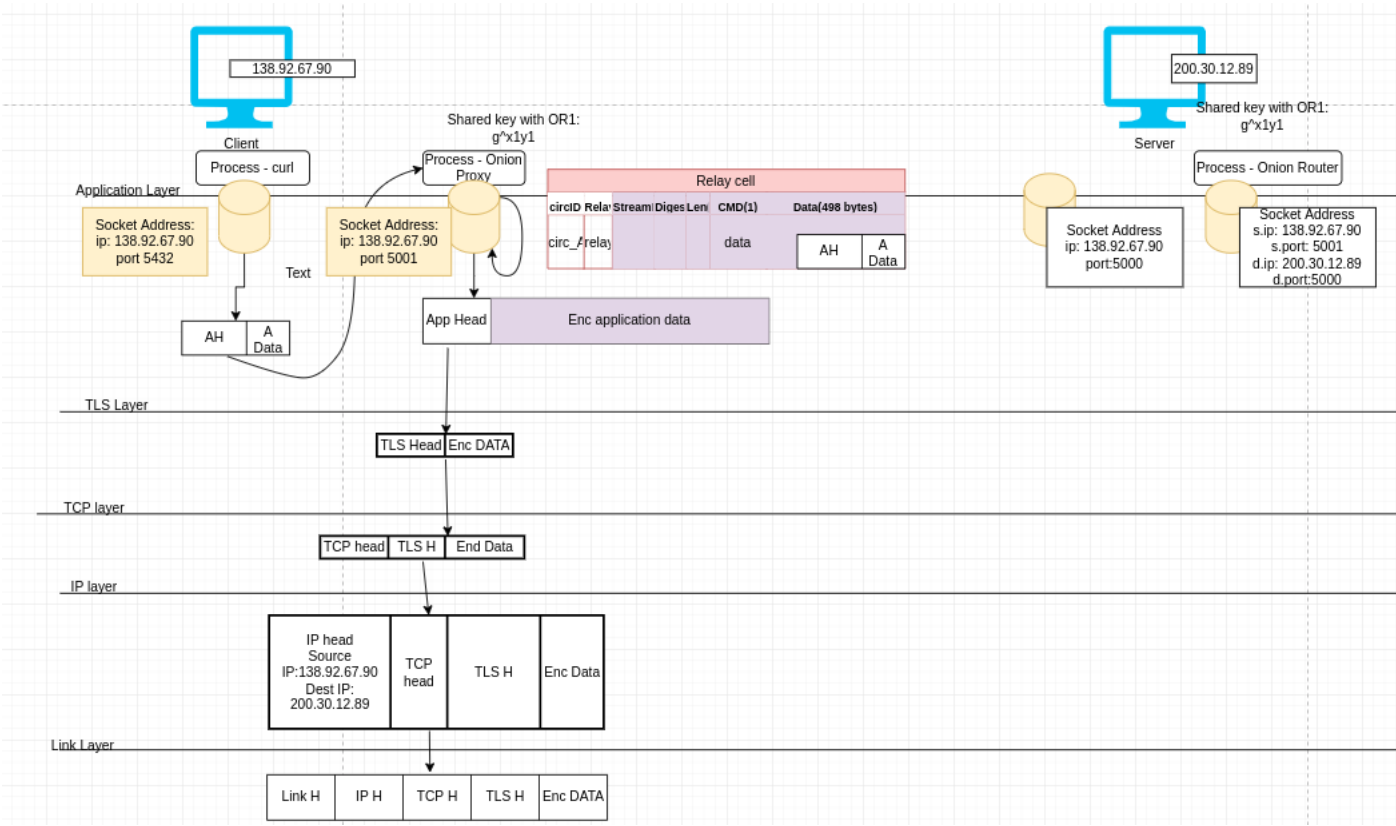


- imagine that one of your LAN machines has two ethernet interfaces (eth0 and eth1) and can therefore act as a gateway between the LAN and the internet.
- We will assume that the rest of the LAN is on the same network as the eth0 interface and that the eth1 interface talks directly the internet. A SOCKS based proxy server installed on the gateway machine can accomplish the following

- The proxy server accepts session requests from SOCKS clients in the LAN on a designated port. If a request does not violate any security policies programmed into the proxy server, the proxy server forwards the request to the internet. Otherwise the request is blocked.
- This property of a proxy server to receive its incoming LAN-side requests for different types of services on a single port and to then forward the requests onwards into the internet to specific ports on specific internet hosts is referred to as port forwarding. Port forwarding is also referred to as tunneling
- Yes, this is exactly like NAT, but what you get with SOCKS is that your IP is hidden even when it's a public address

3.8 (work in progress)Opening and closing TCP streams

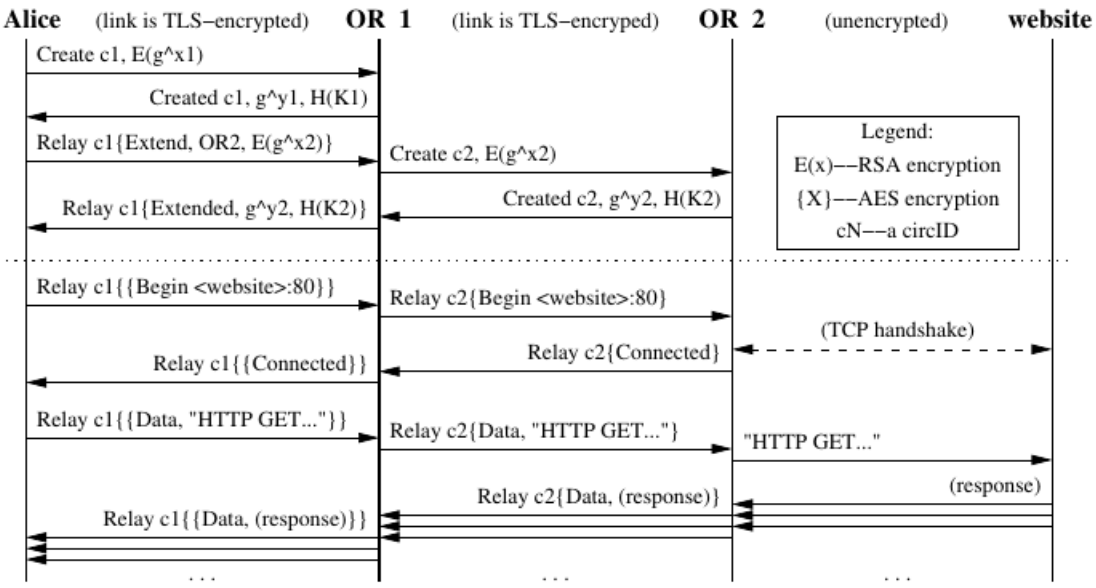
(Figure showing a sockified client application sending network sys calls to OP(socks proxy))



- When Alice's application wants a TCP connection to a given address and port, it asks the OP (via SOCKS) to make the connection.
- The OP chooses the newest open circuit (or cre- ates one if needed), and chooses a suitable OR on that circuit to be the exit node (usually the last node, but maybe others due to exit policy conflicts)

- The OP then opens the stream by sending a relay begin cell to the exit node, using a new random streamID.
- Once the exit node connects to the remote host, it responds with a relay connected cell. Upon receipt, the OP sends a SOCKS reply to notify the application of its success.

The OP now accepts data from the application's TCP stream, packaging it into relay data cells and sending those cells along the circuit to the chosen OR.



4 Towards Goal 2

4.1 Installing and setting up tor, what actually is there in the package

I downloaded tor from the Synaptics Package manager.

```
sudo apt-get install tor
```

installs all three packages listed below:

- tor : tor client. Located at **usr/bin/tor**. Main job of the Tor client /usr/bin/tor is to construct a circuit through the list of relays supplied by one of the Directory Authorities.
- tor-geoipdb :
    1. This package provides a GeoIP database for Tor, i.e. **it maps IPv4 addresses to countries.**

2. Bridge relays (special Tor relays that aren't listed in the main Tor directory) use this information to report which countries they see connections from. These statistics enable the Tor network operators to learn when certain countries start blocking access to bridges.
3. Clients can also use this to learn what country each relay is in, so Tor controllers like arm or Vidalia can use it, or if they want to configure path selection preferences.

- torsocks:

1. SOCKS proxy server: This is Tor SOCKS proxy server. It runs on your machine. By default, the port assigned to this proxy server is 9050. It is this proxy server that will act as the OP (Onion Proxy) in your machine.
2. torsocks[a shell script located at **/usr/bin/torsocks**]You interact with the Tor SOCKS proxy with the shellscript

Seting up tor in Linux:

1. We need to customize the Tor config file that is located at /etc/tor/torrc . This file is going to require a password hash for the password you plan to use in order to limit access to your Tor SOCKS proxy running on your machine

```
tor --hash-password your_password
```

2. edit the file: /etc/tor/torrc, where the xxxxxxxxxxxxxxxx bit is your hashed password

```
Log debug file /var/log/tor/debug.log
ControlPort 9051
HashedControlPassword xxxxxxxxxxxxxxxx
```

3. restart tor

```
sudo /etc/init.d/tor restart
```

4. to verify if tor is running:

```
sudo echo -e 'AUTHENTICATE
"your_password"\r\nsignal NEWNYM\r\nQUIT'| nc 127.0.0.1 9051
```

4.2 Demo

- Run

```
curl https://api.ipify.org
```

- Now we, run the same command with the help of the tor client through the Tor SOCKS proxy running at port 9050 by:

```
c4ndyp0p@pop-os:~$ curl -w "\n" https://api.ipify.org
223.188.249.107
c4ndyp0p@pop-os:~$ torsocks curl -w "\n" https://api.ipify.org
185.220.103.115
```

```
torsocks curl https://api.ipify.org
```

- The shellscript torsocks in the call shown above causes the tor client in your Ubuntu machine at /usr/bin/tor to reach out to a special Tor server known as a Directory Authority for a list of ORs, now more generally referred to as Tor relays. From the list of the relays returned by the Directory Authority, your Tor client constructs a circuit, which typically involves three relays, to the destination IP address. In the example shown above, the destination is the web server at ttps://api.ipify.org .

5 Towards Goal 3:

5.1 Rendezvous points and hidden services:

Motivation:

1. Bob wants to offer a TCP service such as a webserver, called Webservice1.
2. Recall, a TCP service is uniquely define by a two tuple `Host IP, Service Port`
3. Bob somehow wants to host Webservice1 without revealing his Host IP.
4. Anyone in the world, with some identifier(analogous to a url) for Webservice1, should be able to exchange data with it. Without knowing it's IP address.
5. Bob needs a solution?

TCP service should fulfill the following criteria:

1. **Access Control:** Bob needs a way to filter incoming requests, so an attacker cannot flood Bob simply by making many connections to him.
2. **Robustness:** Bob should be able to maintain a long-term pseudonymous identity even in the presence of router failure. Bob's service must not be tied to a single OR, and Bob must be able to migrate his service across ORs.
3. **Smear-resistance:** A social attacker should not be able to "frame" a rendezvous router by offering an illegal or disreputable location-hidden service and making observers believe the router created that service.
4. **Application-transparency:** Although we require users to run special soft-ware to access location-hidden servers, we must not require them to modify their applications.

(work in progress, no citations) How does it work: The following steps are performed on behalf of Alice and Bob by their local OPs; application integration is described more fully below.
Rendevouz spec: https://gitweb.torproject.org/torspec.git/tree/rend-spec.txt?id=9c218f93b9b84587c2b20

1. **Bob configures his local OP.**

   Bob → Bob's OP: "Offer IP:Port as public-key-name:Port".

2. **Bob's OP establishes his introduction points.**

   The first time the OP provides an advertised service, it generates a public/private keypair (stored locally).

   "This public key is (currently) associated to me."

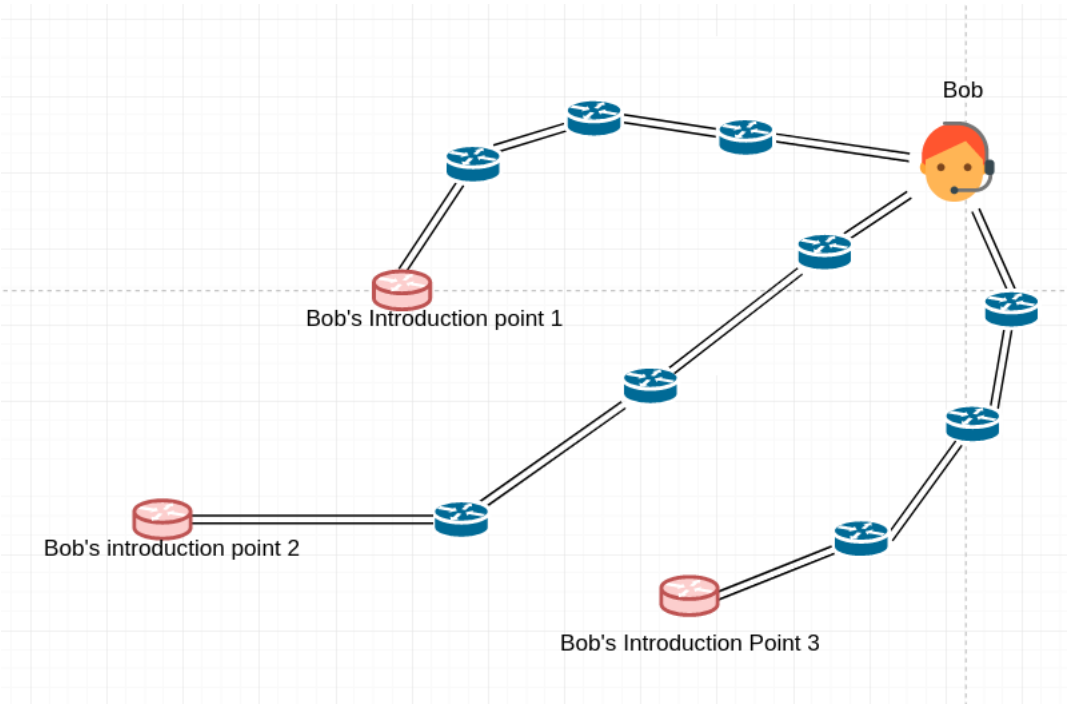
   (a) The OP chooses a small number of Tor servers as introduction points. The OP establishes a new introduction circuit to each introduction point. These circuits MUST NOT be used for anything but hidden service introduction. To establish the introduction, Bob sends a *RELAY_COMMAND_ESTABLISH_INTRO* cell, containing:

```
KL   Key length                              [2 octets]
PK   Bob's public key or service key         [KL octets]
HS   Hash of session info                    [20 octets]
SIG  Signature of above information          [variable]
```

```
KL is the length of PK, in octets.

To prevent replay attacks, the HS field contains a SHA-1 hash based on the
shared secret KH between Bob's OP and the introduction point, as
follows:
    HS = H(KH | "INTRODUCE")
That is:
    HS = H(KH | [49 4E 54 52 4F 44 55 43 45])
(KH, as specified in tor-spec.txt, is H(g^xy | [00]) .)
```

   (b) Upon receiving such a cell, the OR first checks that the signature is correct with the included public key. If so, it checks whether HS is correct given the shared state between Bob's OP and the OR. If either check fails, the OP discards the cell; otherwise, it associates the circuit with Bob's public key, and dissociates any other circuits currently associated with PK. On success, the OR sends Bob a *RELAY_COMMAND_INTRO_ESTABLISHED* cell with an empty payload.



3. **Bob's OP generates service descriptors**

```
The "V0" descriptor contains:

  KL    Key length                          [2 octets]
  PK    Bob's public key                    [KL octets]
  TS    A timestamp                         [4 octets]
  NI    Number of introduction points       [2 octets]
  Ipt   A list of NUL-terminated ORs        [variable]
  SIG   Signature of above fields           [variable]

TS is the number of seconds elapsed since Jan 1, 1970.
```

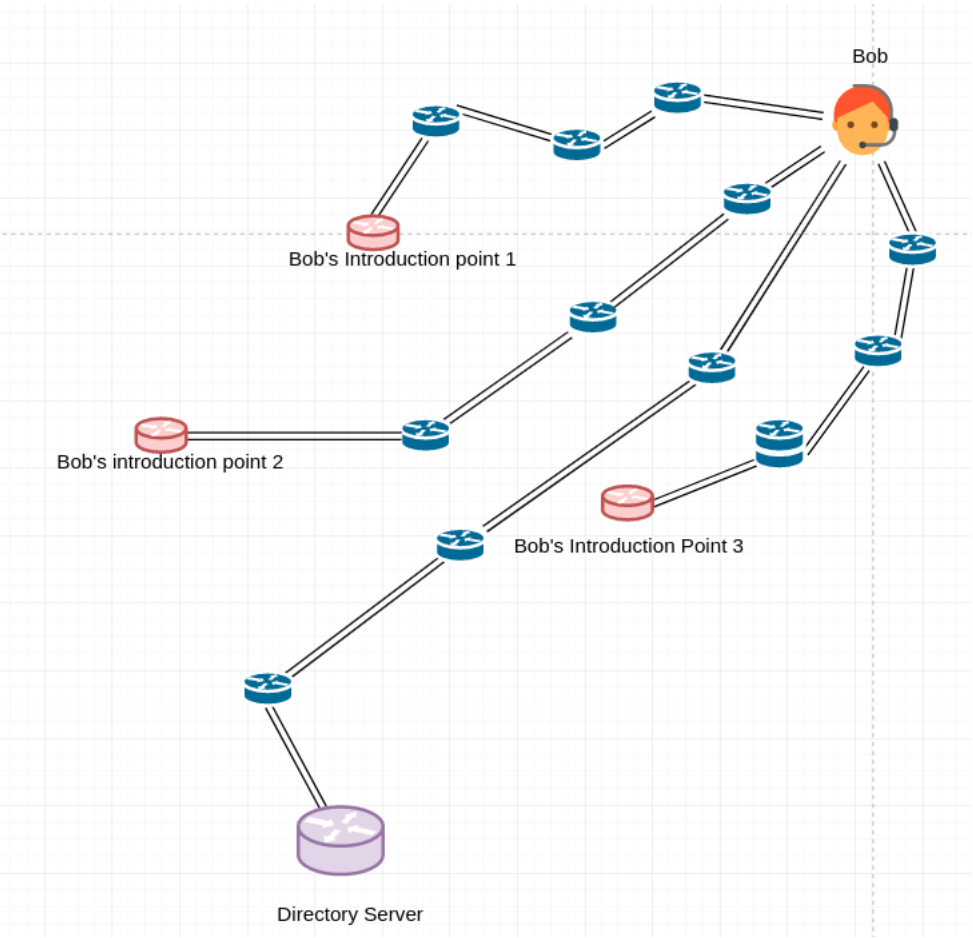
4. **Bob's OP → directory service via Tor: publishes/advertises Bob's service descriptor**

```
[advertisement]
"Meet public-key X at introduction point A, B, or C." (signed)
```

   (a) Bob's OP advertises his service descriptor to a fixed set of v0 hidden service directory servers

   (b) Upon receiving a descriptor, the directory server checks the signature, and discards the descriptor if the signature does not match the enclosed public key. Next,

the directory server checks the timestamp. If the timestamp is more than 24 hours in the past or more than 1 hour in the future, or the directory server already has a newer descriptor with the same public key, the server discards the descriptor. Otherwise, the server discards any older descriptors with the same public key and version format, and associates the new descriptor with the public key. The directory server remembers this descriptor for at least 24 hours after its timestamp. At least every 18 hours, Bob's OP uploads a fresh descriptor.



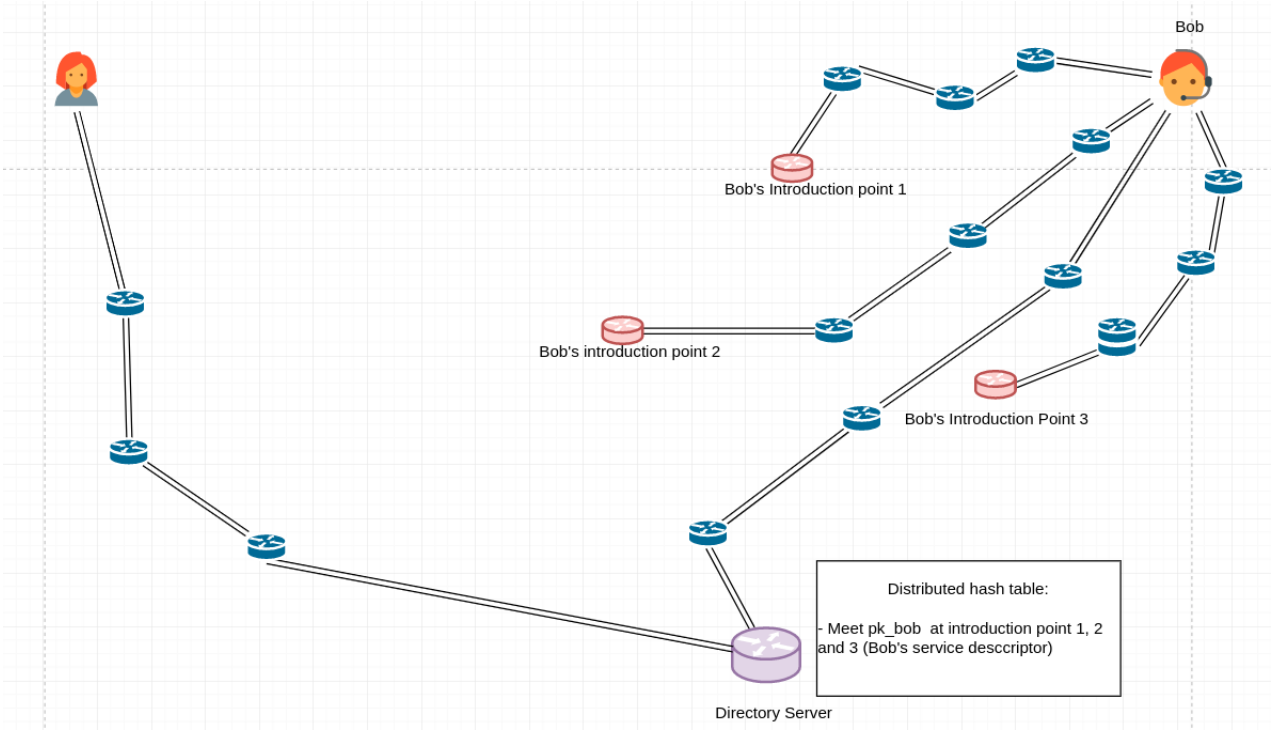
5. **Out of band, Alice receives a z.onion:port address. She opens a SOCKS connection to her OP, and requests z.onion:port.** When Alice receives a pointer to a location-hidden service, it is as a hostname of the form "z.onion", where z is a base32 encoding of a 10-octet hash of Bob's service's public key, computed as follows:

   (a) Let H = H(PK).

   (b) Let H' = the first 80 bits of H, considering each octet from most significant bit to least significant bit.

   (c) Generate a 16-character encoding of H', using base32 as defined in RFC 4648.

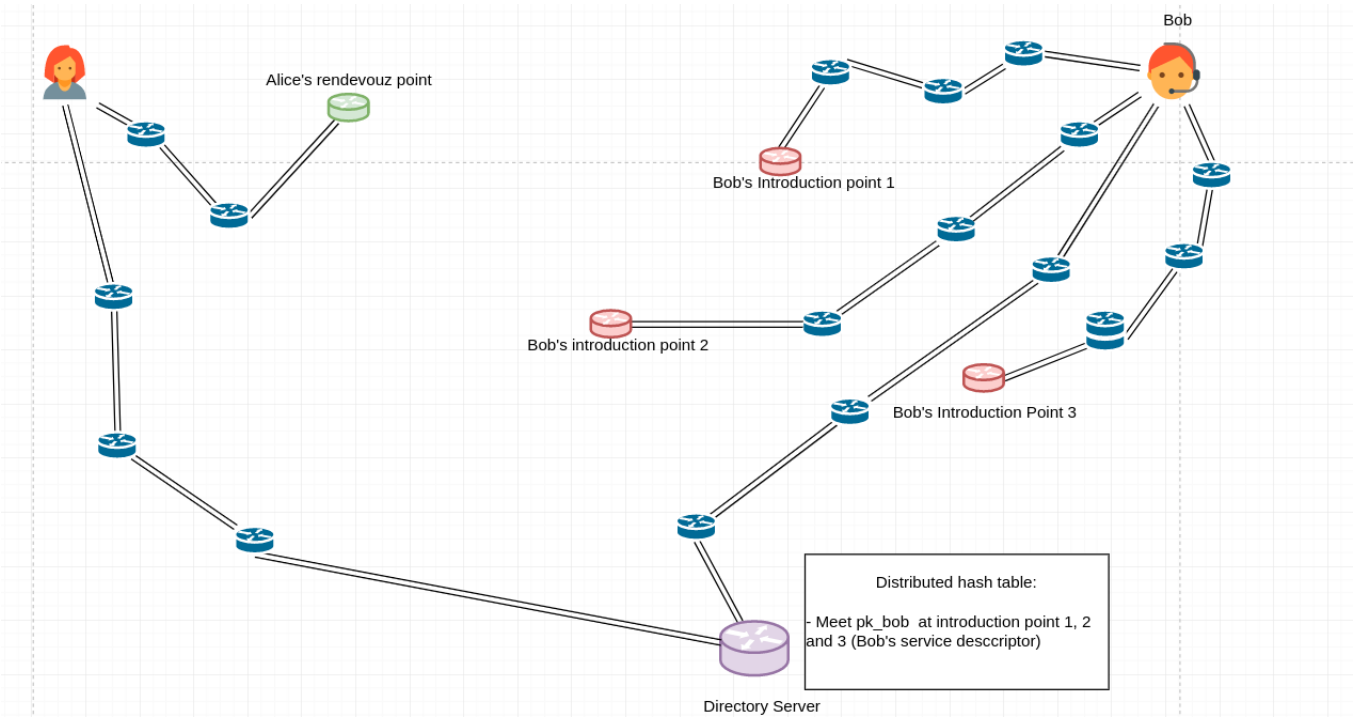
6. Alice's OP retrieves Bob's descriptor via Tor. [descriptor lookup.]



(a) Alice's OP fetches the service descriptor from the fixed set of v0 hidden service directory servers

(b) Upon receiving a service descriptor, Alice verifies with the same process as the directory server uses

7. Alice's OP chooses a rendezvous point, opens a circuit to that rendezvous point, and establishes a rendezvous circuit, gives it a randomly chosen "rendezvous cookie" to recognize Bob. [rendezvous setup.]



(a) It does this by establishing a circuit to a randomly chosen OR, and sending a $RELAY-COMMAND-ESTABLISH-RENDEZVOUS$ cell to that OR. The body of that cell contains:

```
RC  Rendezvous cookie
    [20 octets]
```

```
The rendezvous cookie is an arbitrary 20-byte value, chosen randoml
Alice's OP. Alice SHOULD choose a new rendezvous cookie for each new
connection attempt.
```

The rendezvous cookie is an arbitrary 20-byte value, chosen randomly by Alice's OP.

(b) Upon receiving a $RELAY-COMMAND-ESTABLISH-RENDEZVOUS$ cell, the OR associates the RC with the circuit that sent it. It replies to Alice with an empty $RELAY-COMMAND-RENDEZVOUS-ESTABLISHED$ cell to indicate success.

(c) Alice's OP MUST NOT use the circuit which sent the cell for any purpose other than rendezvous with the given location-hidden service.

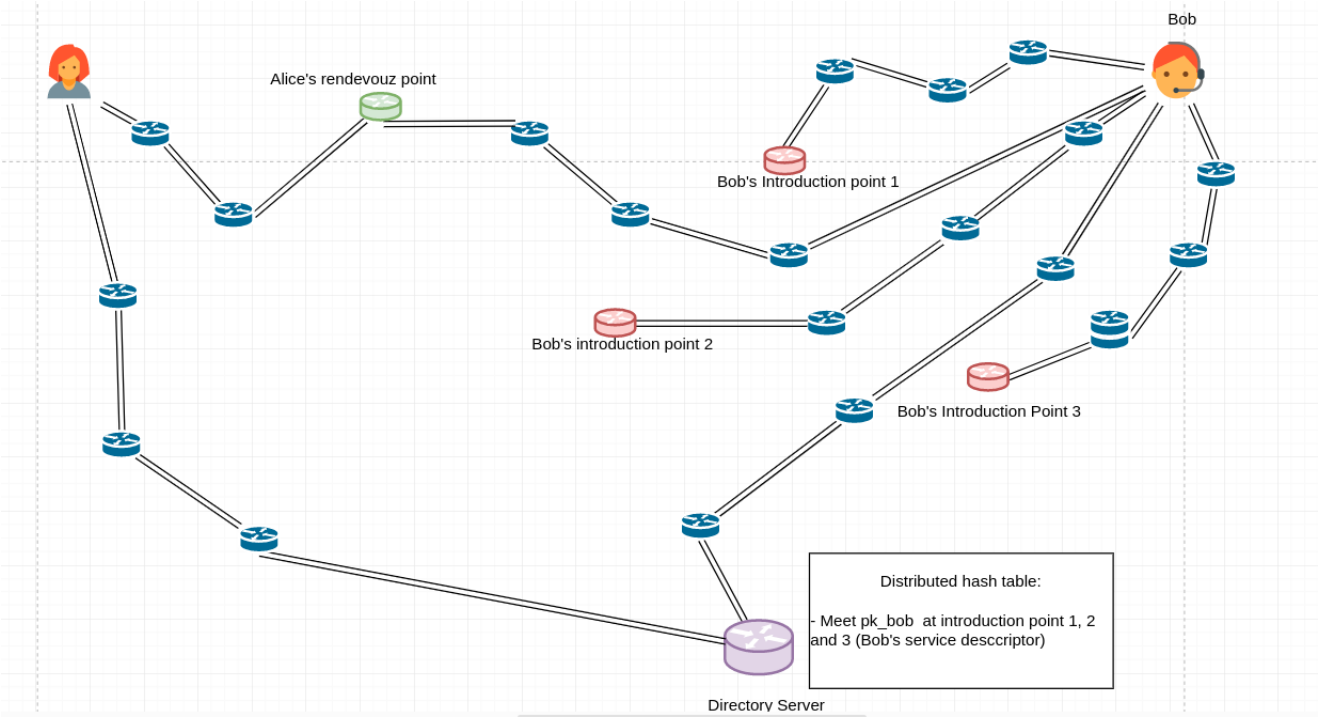
8. **Alice → Introduction point via Tor, and tells it about her rendezvous point. (Encrypted to Bob's Public Key) , telling it about herself, her RP, and rendezvous cookie, and start of a DH handshake. [Introduction 1]**

Alice builds a separate circuit to one of Bob's chosen introduction points, and sends it a $RELAY-COMMAND-INTRODUCE1$ cell containing:

```
Cleartext
   PK_ID  Identifier for Bob's PK      [20 octets]
Encrypted to Bob's PK: (in the v0 intro protocol)
   RP     Rendezvous point's nickname  [20 octets]
   RC     Rendezvous cookie            [20 octets]
   g^x    Diffie-Hellman data, part 1 [128 octets]
```

9. The Introduction point passes this on to Bob's OP via Tor, along the introduction circuit. [Introduction 2]

10. Bob's OP decides whether to connect to Alice, and if so, creates a circuit to Alice's RP via Tor. Establishes a shared circuit.(By sending the rendezvous cookie, the second half of the DH handshake, a hash of the session key they now share)



Alice knows she shares the key only with Bob.

11. **The rendezvous point connects Alice's circuit to Bob's**The Rendezvous point forwards Bob's confirmation to Alice's OP.

    Note that the RP cannot recognise Alice, N

12. Alice's OP sends begin cells to Bob's OP. [Connection]

    and tells them to wait for requests

5.2 Hosting a hidden service:

The tor website provides with brief instructions on how to get a hidden service running at https://community.torproject.org/onion-services/setup/. This guide was also helpful https://www.metaha

tor-hidden-services/

1. Get a working Tor: I setup tor on my server as mentioned above
2. Get a web server working: I used nginx for this. You can visit the page at http://149.102.156.110/
3. Configure your Tor Onion Service: You will need to add the following two lines to your torrc file:

   ```
   HiddenServiceDir /var/lib/tor/my_website/
   HiddenServicePort 80 127.0.0.1:80
   ```

   The HiddenServiceDir line specifies the directory which should contain information and cryptographic keys for your Onion Service. You will want to change the HiddenServiceDir line, so that it points to an actual directory that is readable/writeable by the user that will be running Tor.

   The HiddenServicePort line specifies a virtual port (that is, the port that people visiting your Onion Service will be using), and in the above case it says that any traffic incoming to port 80 of your Onion Service should be redirected to 127.0.0.1:80 (which is where the web server from step 1 is listening).

4. Restart tor

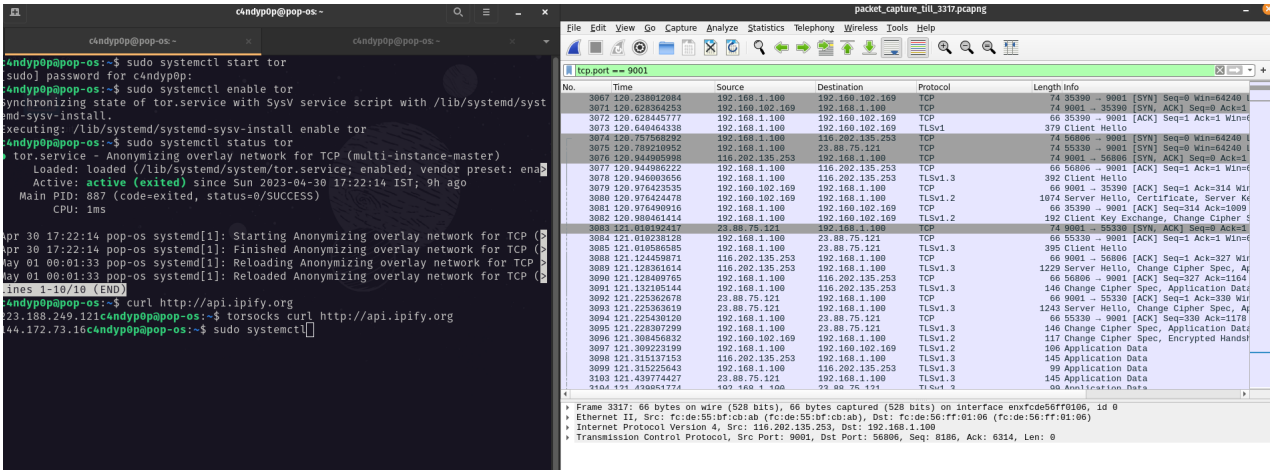
   ```
   sudo systemctl restart tor
   ```

5. Test the Onion Service: Now to get your Onion Service address, go to your HiddenServiceDir directory, and find a file named hostname. The hostname file in your Onion Service configuration directory contains the hostname for your new onion v3 service. Our page is at http://qkazvrlyywk3ufigpji3jw43toxjguir2ledxkizhdc2ylrobcxlbtqd.onion/

6 Towards Goal 4: capture tor packets in wireshark

I was able to isolate some tor packets from my normal traffic by filtering for packets on tcp port 9001. As the defalt port for Onion Routers(running on relay nodes) is 9001.

```
tcp.port == 9001
```



However, right from the beginning all tor traffic is TLS encrypted so I cannot with certainly point out which packets are used for constructing circuits and which are used for sending data, so on and so forth. A lot of research has been done and is ongoing regarding detecting and fingerprinting tor's traffic. These techniques are mainly used by censors and firewalls to block tor.

7 Towards Goal 7:

Current tor source code [6] and spec [8] are on a privately hosted Gitlab instance. To get access, make an account and request for it. Incase of problems, contact a maintainer/volunteer on TRC to get your account approved. I have access to it.

The current documentation seems to be located at: `https://tpo.pages.torproject.net/core/doc/tor/intro.html` [7]

References

[1] The Tor Project. `https://www.torproject.org/`, n.d. Accessed on 2 May 2023.

[2] Tor metrics: Relay and bridge network size. `https://metrics.torproject.org/networksize.html`, n.d. Accessed on 2 May 2023.

[3] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[4] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 21–21, 2004.

[5] Subhash Kak. Computer security lectures. `https://engineering.purdue.edu/kak/compsec/NewLectures/`, n.d. Accessed on 2 May 2023.

[6] Tor Project. Tor project- source code. `https://gitlab.torproject.org/`, n.d.

[7] Tor Project. Tor project documentation. `https://tpo.pages.torproject.net/core/doc/tor/intro.html`, n.d. Accessed on 3 May 2023.

[8] Tor Project. Tor spec. `https://gitlab.torproject.org/tpo`, n.d.

[9] Wikipedia contributors. Mix network — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Mix_network`, 2021. Accessed on 2 May 2023.

[10] Diwen Xue, Reethika Ramesh, Arham Jain, Michalis Kallitsis, J. Alex Halderman, Jedidiah R. Crandall, and Roya Ensafi. OpenVPN is open to VPN fingerprinting. In *Proceedings of the 31st USENIX Security Symposium*. USENIX Association, 2022.