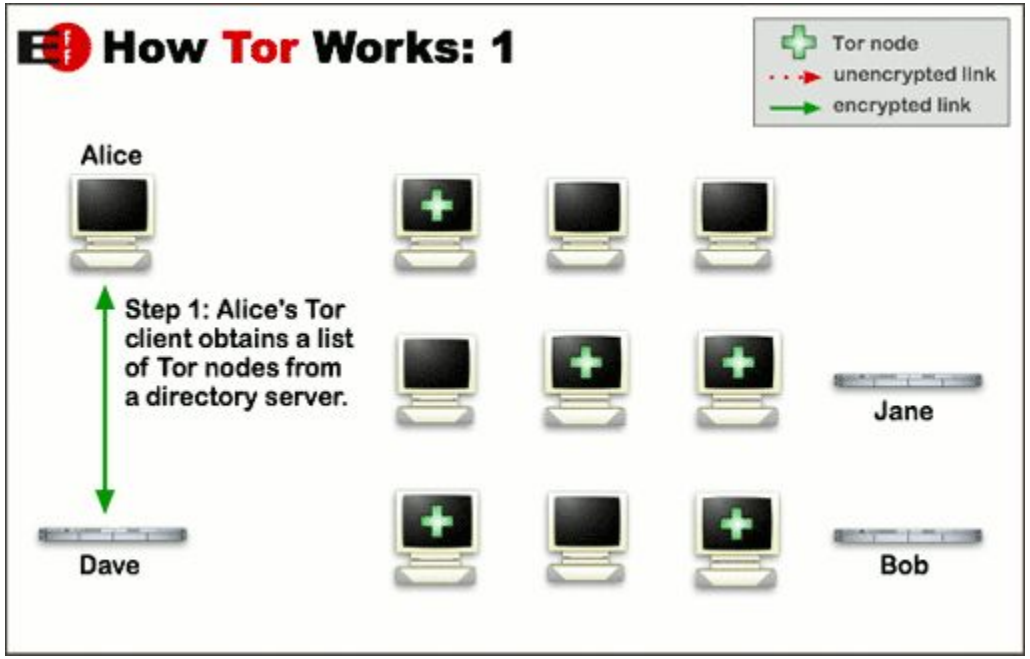
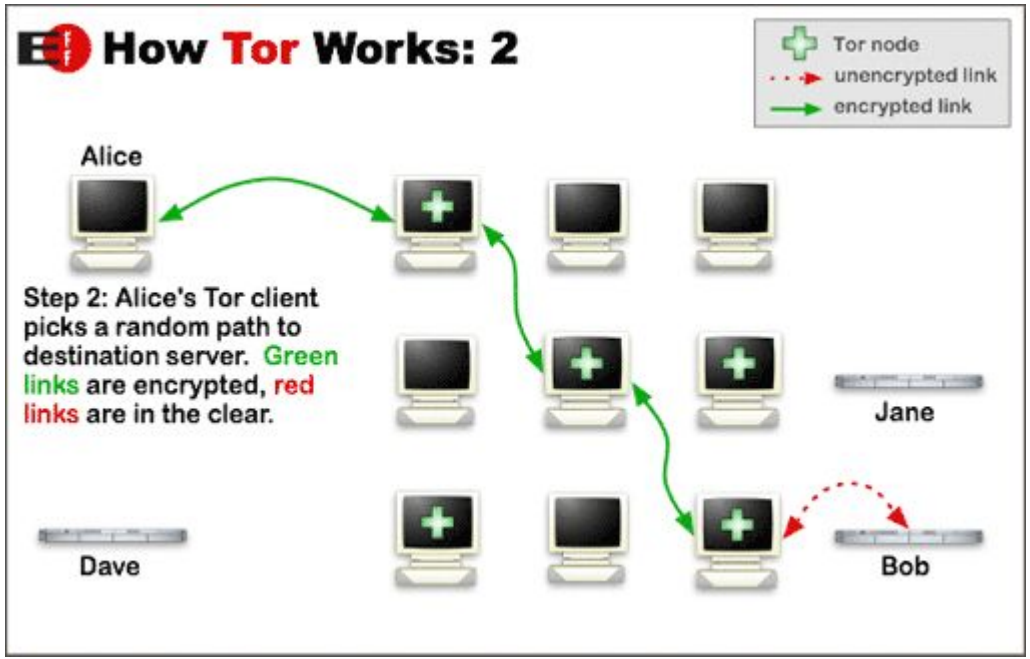
Tor operations (1)

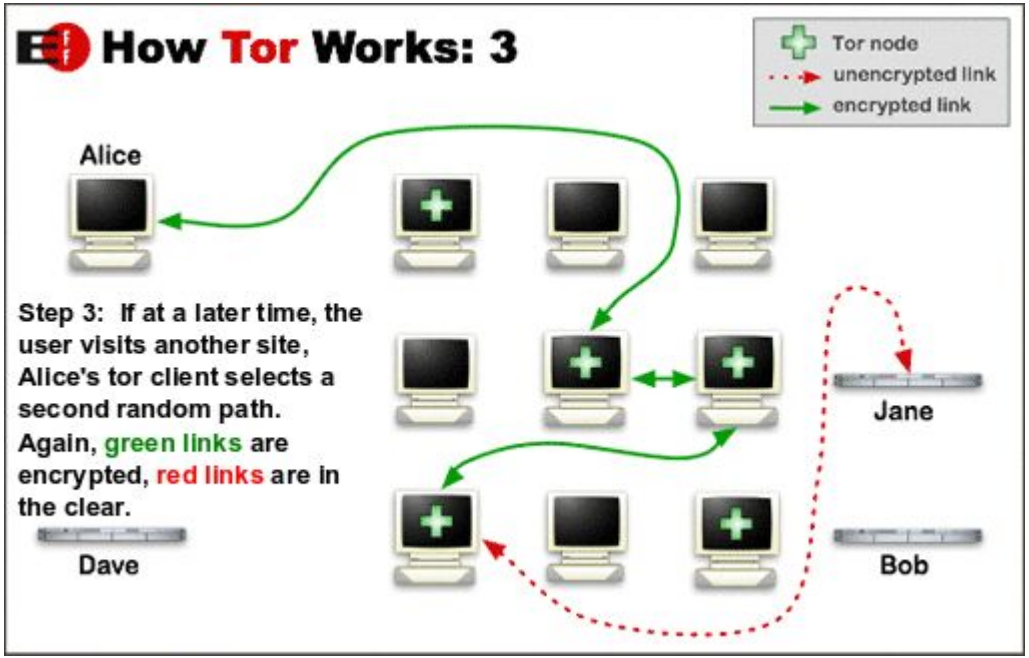


Source: https://2019.www.torproject.org/about/overview.html.en

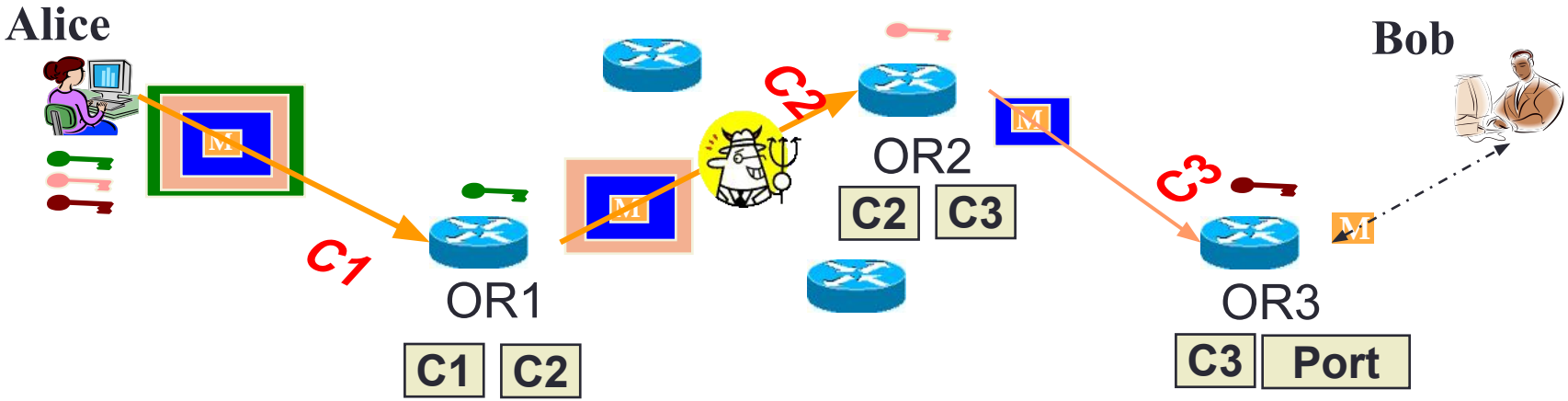
Tor operations (2)



Tor operations (3)

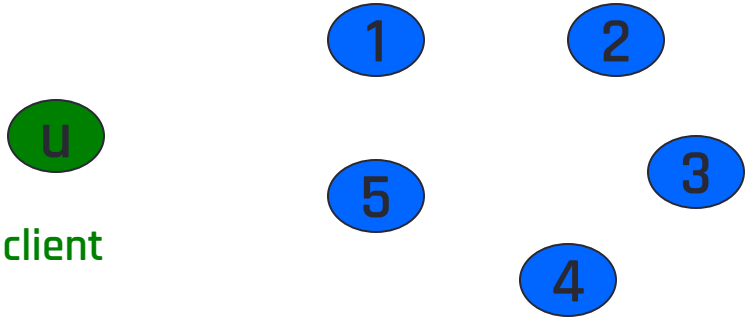


Onion Routing



- A circuit is built incrementally one hop by one hop
- Onion-like encryption
  - Alice negotiates an AES key with each router
  - Messages are divided into equal sized cells
  - Each router knows only its predecessor and successor
  - Only the Exit router (OR3) can see the message, however it does not know where the message is from

Onion Routing

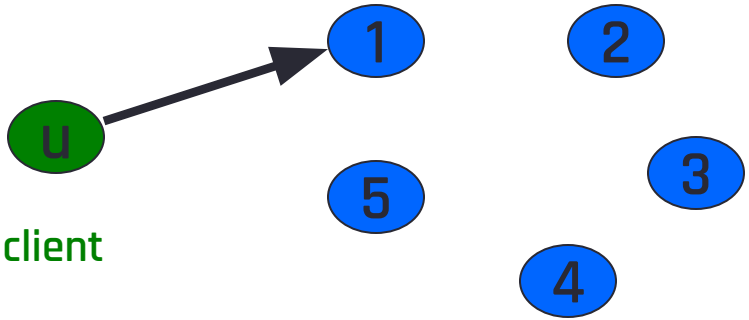


User *u* running client

Routers running servers

Internet destination *d*

Onion Routing



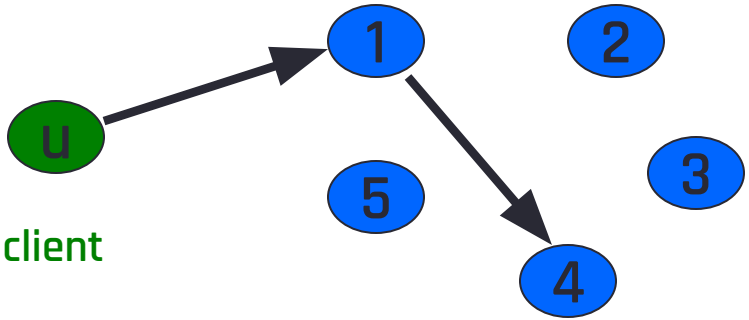
User *u* running client

Routers running servers

Internet destination *d*

1. *u* creates l-hop circuit through routers

Onion Routing



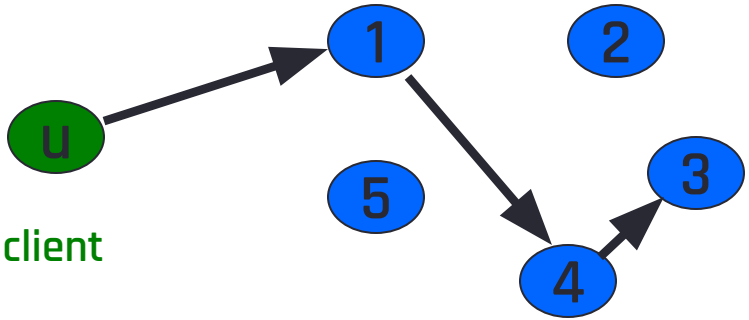
User *u* running client

Internet destination *d*

Routers running servers

1. *u* creates l-hop circuit through routers

Onion Routing



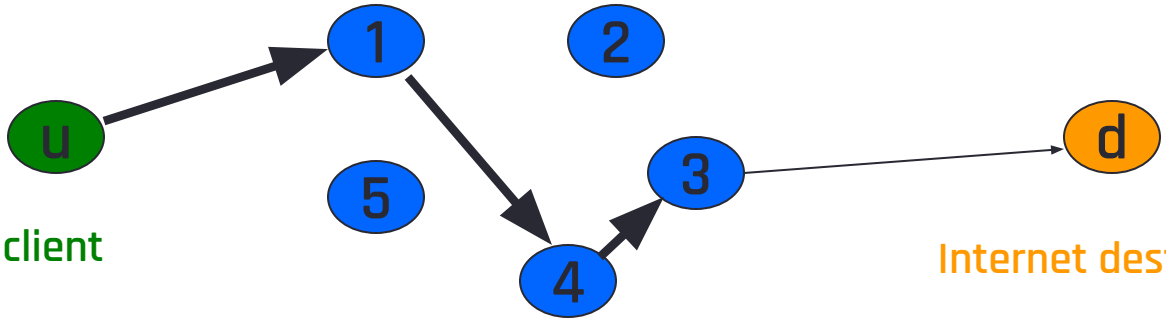
User *u* running client

Routers running servers

Internet destination *d*

1. *u* creates l-hop circuit through routers

Onion Routing



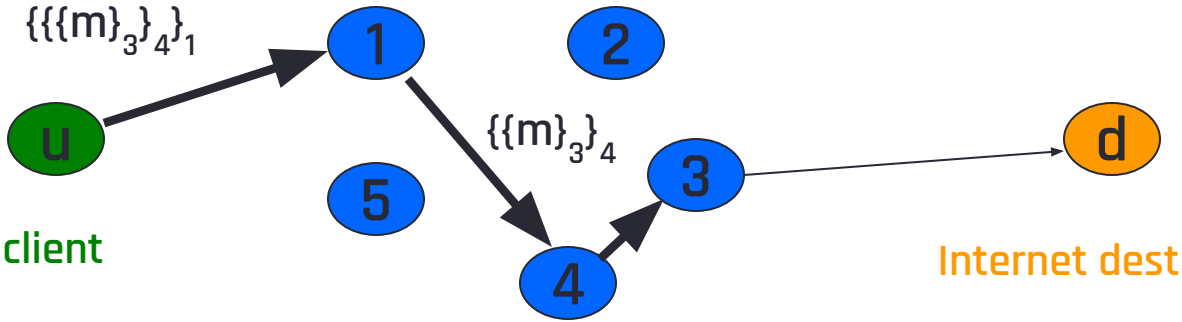
User *u* running client

Internet destination *d*

Routers running servers

1. u creates l-hop circuit through routers
2. u opens a stream in the circuit to d

Onion Routing



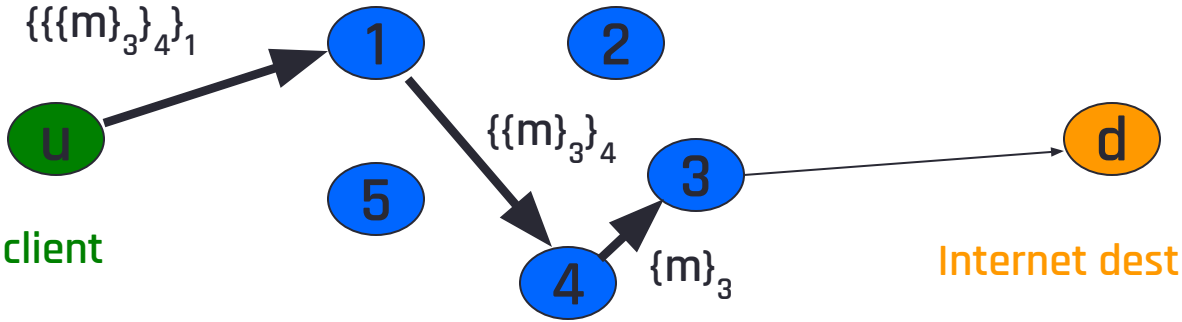
User *u* running client

Internet destination *d*

Routers running servers

1. u creates l-hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged

Onion Routing



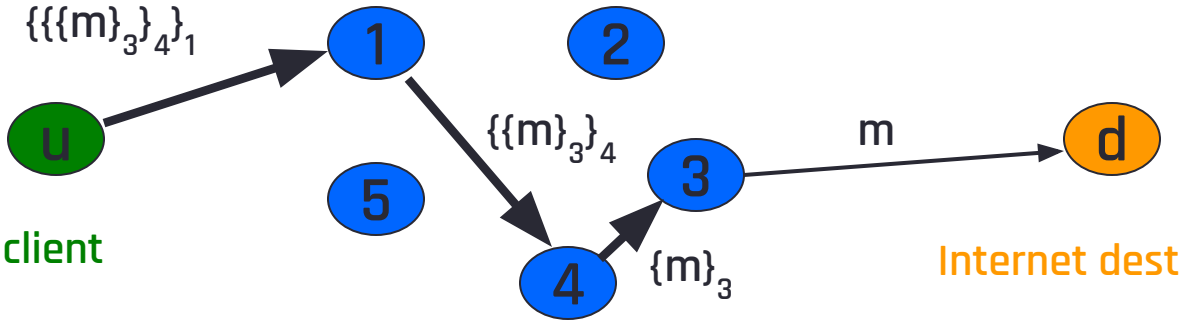
User *u* running client

Internet destination *d*

Routers running servers

1. u creates l-hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged

Onion Routing



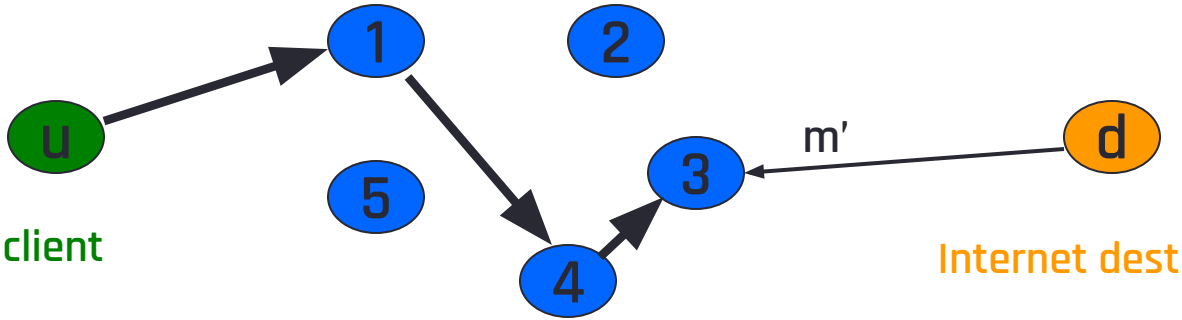
User *u* running client

Internet destination *d*

Routers running servers

1. u creates l-hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged

Onion Routing



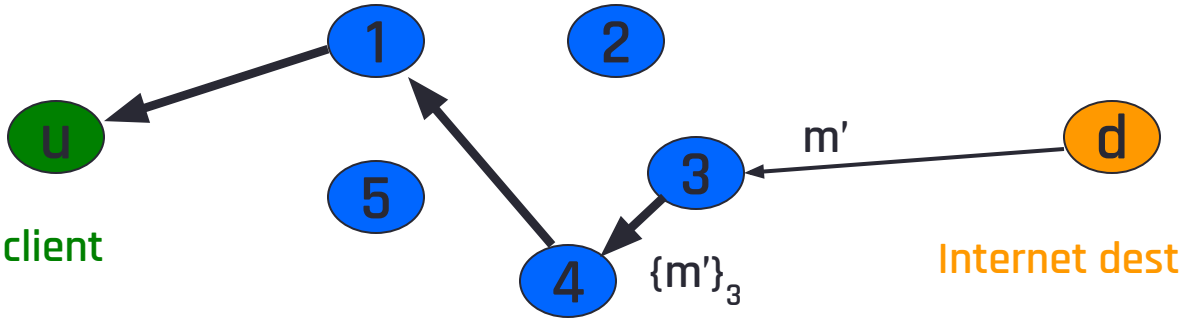
User *u* running client

Internet destination *d*

Routers running servers

1. u creates l-hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged

Onion Routing



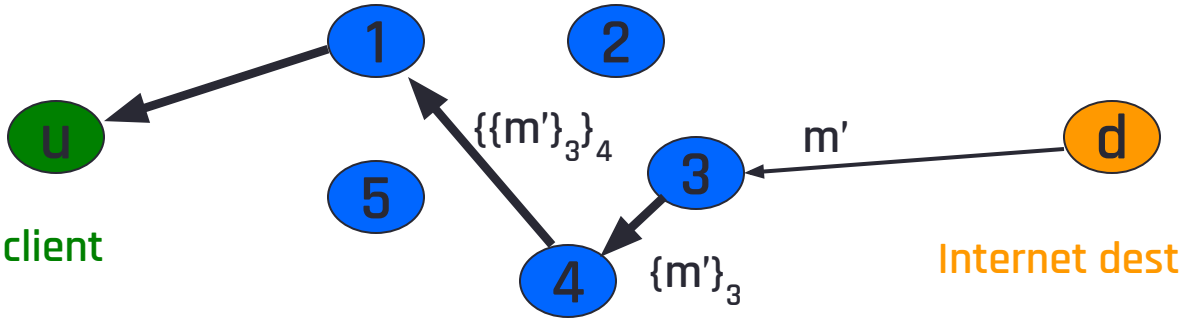
User *u* running client

Internet destination *d*

Routers running servers

1. u creates l-hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged

Onion Routing



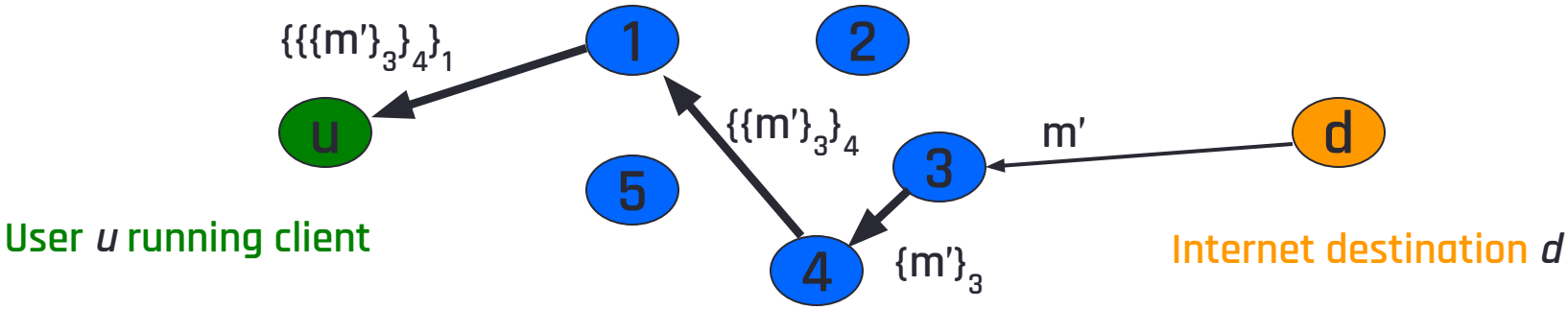
User *u* running client

Internet destination *d*

Routers running servers

1. u creates l-hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged

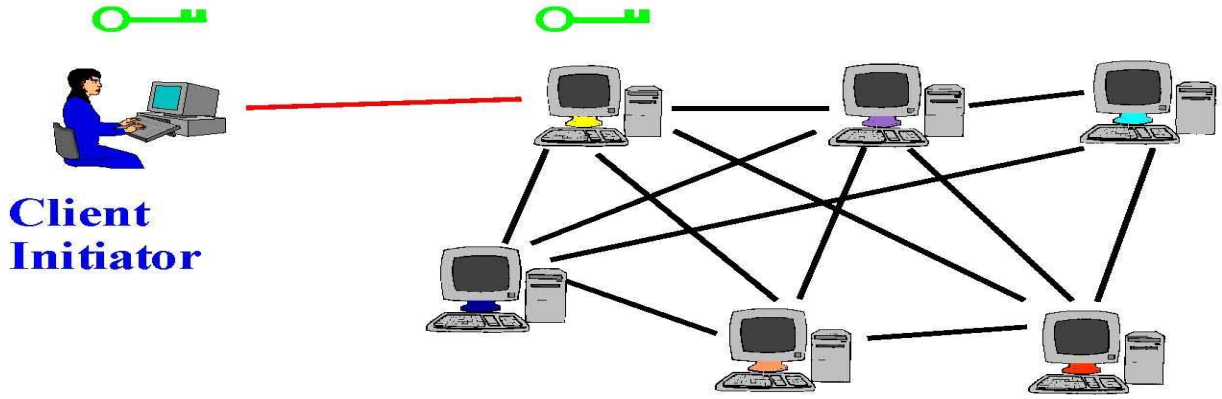
Onion Routing



1. u creates l-hop circuit through routers
2. u opens a stream in the circuit to d
3. Data are exchanged
4. Stream is closed.

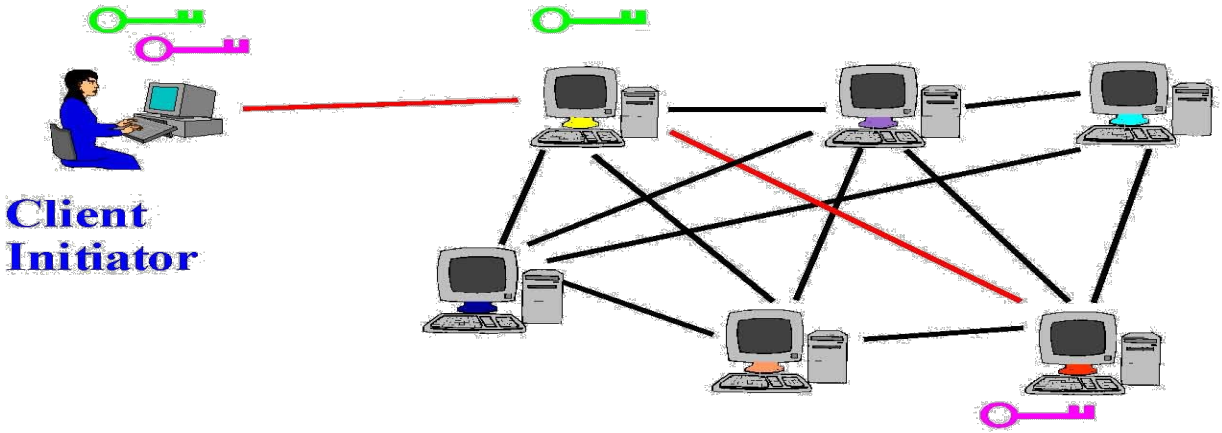
Tor Circuit Setup (1)

- Client proxy establish a symmetric session key and circuit with relay node #1



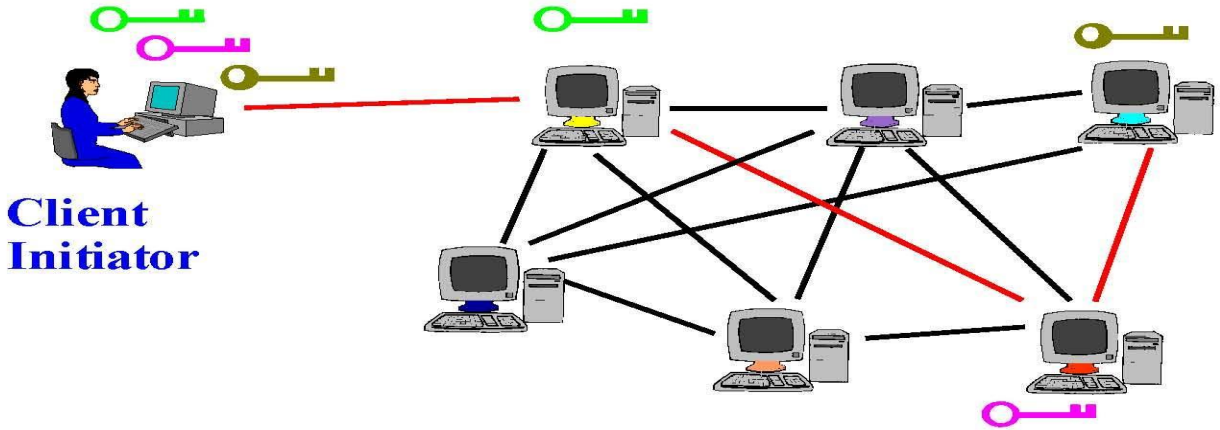
Tor Circuit Setup (2)

- Client proxy extends the circuit by establishing a symmetric session key with relay node #2
    - Tunnel through relay node #1 - don't need  !



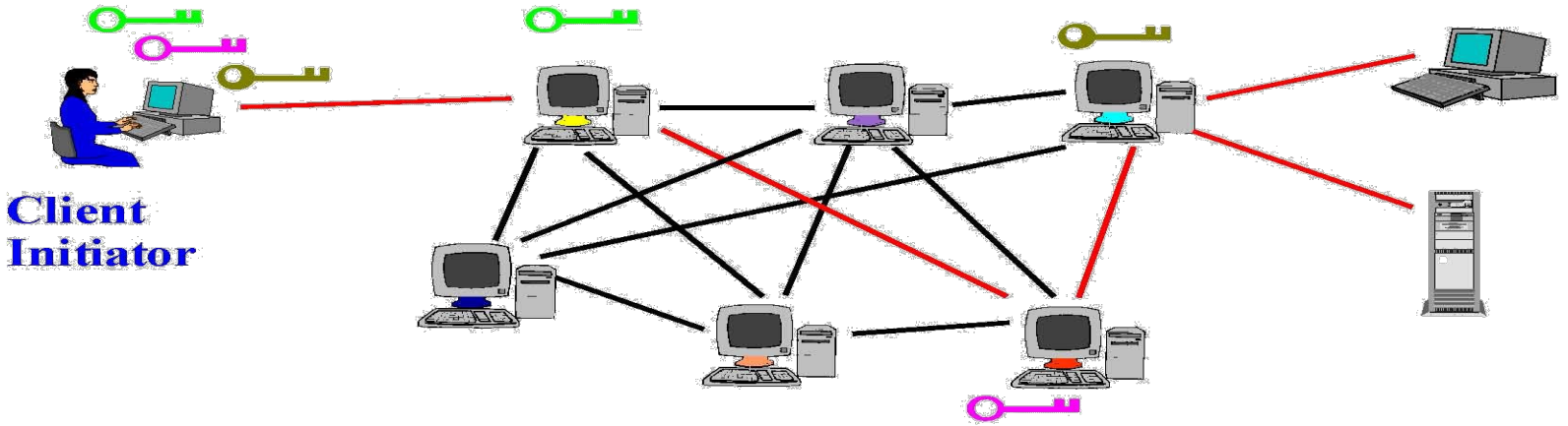
Tor Circuit Setup (3)

- Client proxy extends the circuit by establishing a symmetric session key with relay node #3
  - Tunnel through relay nodes #1 and #2



Using a Tor Circuit

- Client applications connect and communicate over the established Tor circuit
  - Datagrams decrypted and re-encrypted at each link

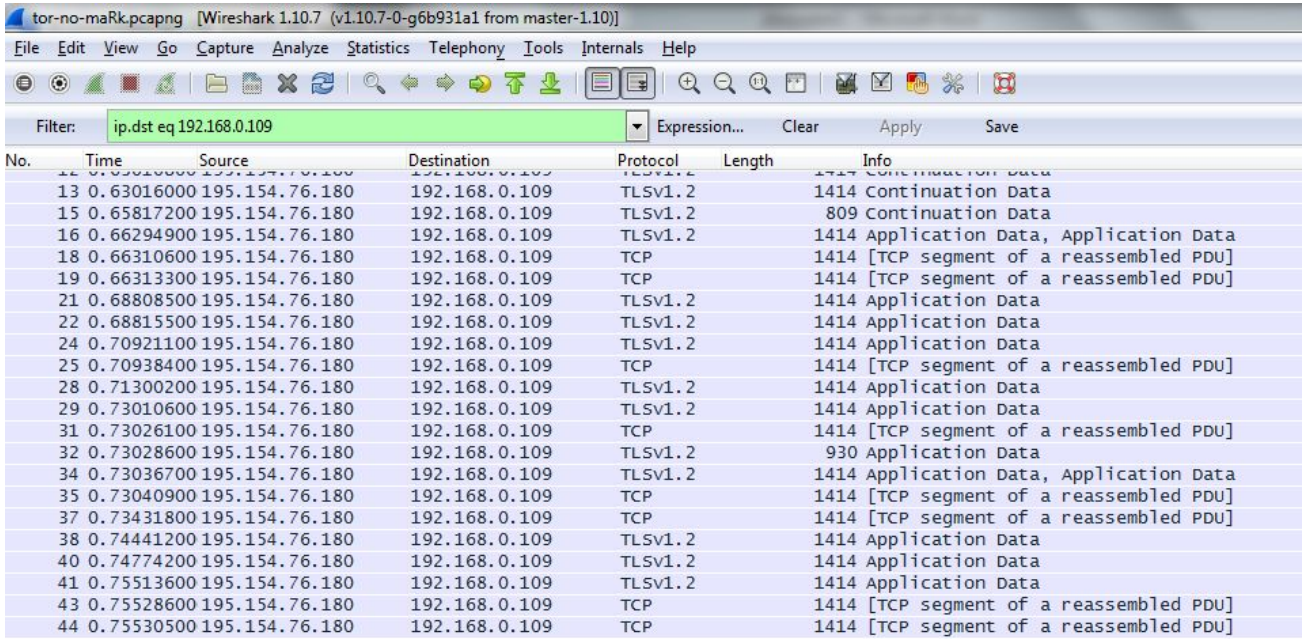


Design

- Overlay network on the user level
- Onion Routers (OR) route traffic
- Onion Proxy (OP) fetches directories and creates virtual circuits on the network on behalf of users.
- Uses TCP with TLS
- All data is sent in fixed size (bytes) cells

How does Tor traffic look like?

- Snippet of traffic observed at entry node (Guard OR)



No.	Time	Source	Destination	Protocol	Length	Info
13	0.63016000	195.154.76.180	192.168.0.109	TLSv1.2	1414	Continuation Data
15	0.65817200	195.154.76.180	192.168.0.109	TLSv1.2	809	Continuation Data
16	0.66294900	195.154.76.180	192.168.0.109	TLSv1.2	1414	Application Data, Application Data
18	0.66310600	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
19	0.66313300	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
21	0.68808500	195.154.76.180	192.168.0.109	TLSv1.2	1414	Application Data
22	0.68815500	195.154.76.180	192.168.0.109	TLSv1.2	1414	Application Data
24	0.70921100	195.154.76.180	192.168.0.109	TLSv1.2	1414	Application Data
25	0.70938400	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
28	0.71300200	195.154.76.180	192.168.0.109	TLSv1.2	1414	Application Data
29	0.73010600	195.154.76.180	192.168.0.109	TLSv1.2	1414	Application Data
31	0.73026100	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
32	0.73028600	195.154.76.180	192.168.0.109	TLSv1.2	930	Application Data
34	0.73036700	195.154.76.180	192.168.0.109	TLSv1.2	1414	Application Data, Application Data
35	0.73040900	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
37	0.73431800	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
38	0.74441200	195.154.76.180	192.168.0.109	TLSv1.2	1414	Application Data
40	0.74774200	195.154.76.180	192.168.0.109	TLSv1.2	1414	Application Data
41	0.75513600	195.154.76.180	192.168.0.109	TLSv1.2	1414	Application Data
43	0.75528600	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]
44	0.75530500	195.154.76.180	192.168.0.109	TCP	1414	[TCP segment of a reassembled PDU]

Additional functionality

- Integrity checking
  - Only done at the edges of a stream
  - SHA-1 digest of data sent and received
  - First 4 bytes of digest are sent with each message for verification
- OR-to-OR congestion might happen if too many users choose the same OR-to-OR connection.
- Circuit Level throttling
  - 2 windows keep track of relay data to be transmitted to other ORs (packaging window) and data transmitted out of the network (delivery window)
  - Windows are decremented after forwarding packets and increments on a relay sendme message towards OP with streamID zero.
  - When a window reaches 0, no messages are forwarded

Using Tor

- Many applications can share one circuit
  - Multiple TCP streams over one anonymous connection
- Tor router doesn't need root privileges
  - Encourages people to set up their own routers
  - More participants = better anonymity for everyone
- Directory servers
  - Maintain lists of active relay nodes, their locations, current public keys, etc.
  - Control how new nodes join the network
    - "Sybil attack": attacker creates a large number of relays
  - Directory servers' keys ship with Tor code

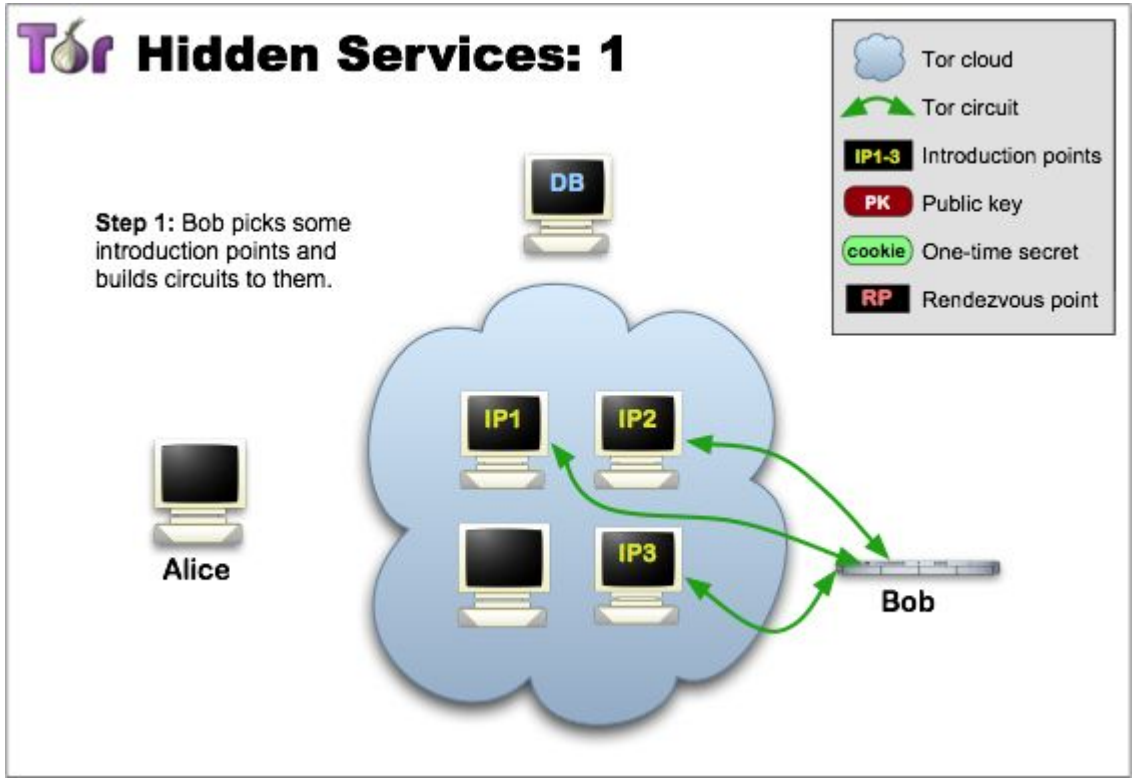
Hidden Services

- Goal: deploy a server on the Internet that anyone can connect to without knowing where it is or who runs it
- Accessible from anywhere
- Resistant to censorship, denial of service, physical attack
  - Network address of the server is hidden, thus can't find the physical server

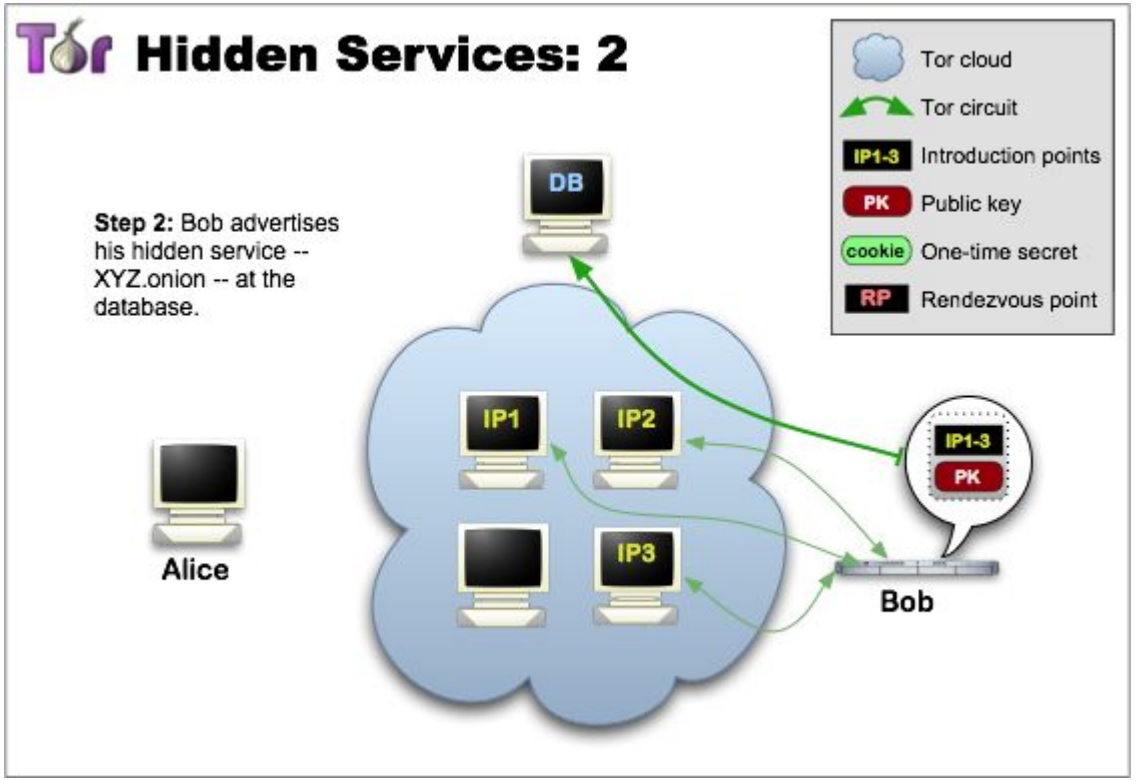
Hidden Service and Rendezvous Points

- Location-hidden services allow Bob to offer a TCP service without revealing his IP address.
- Tor accommodates receiver anonymity by allowing location hidden services
- Design goals for location hidden services
  - Access Control: filtering incoming requests (against flooding)
  - Robustness: maintain a long-term pseudonymous identity (ability to migrate service across Ors)
  - Smear-resistance: social attacker should not be able to “frame” a rendezvous router by offering an illegal location-hidden service and making observers believe the router created that service
  - Application transparency: same unmodified application
- Location hidden service leverage rendezvous points

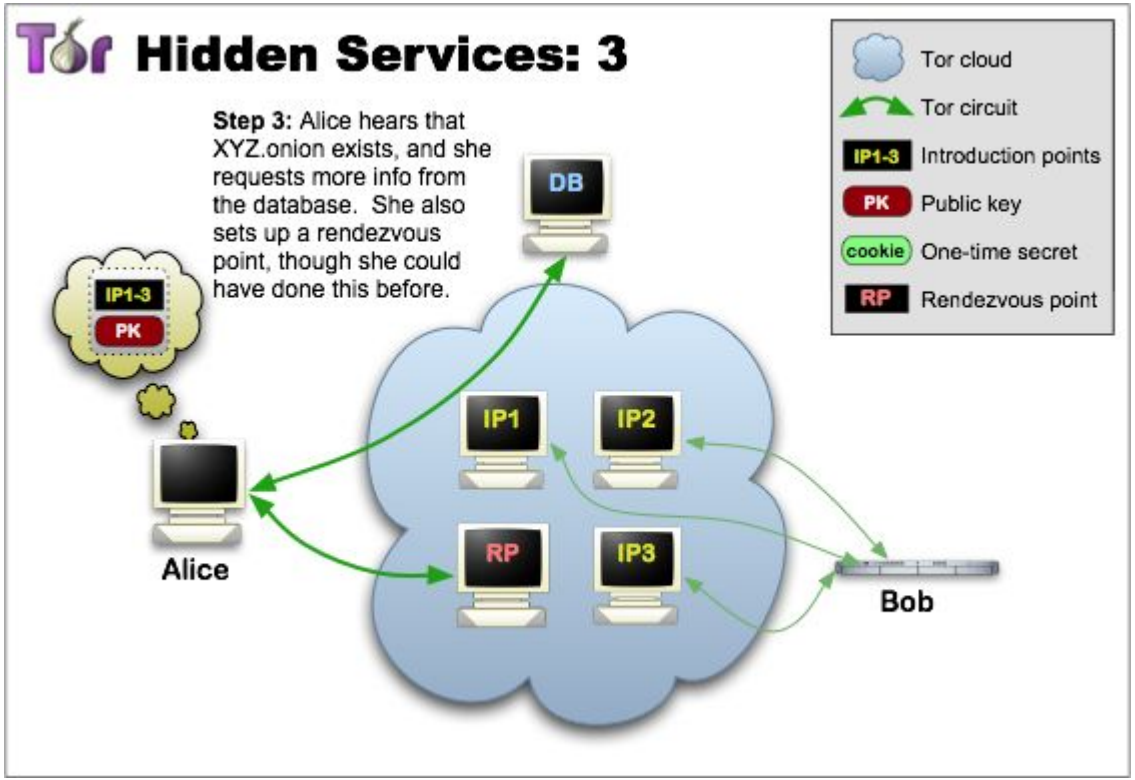
Setting up a hidden service (1)



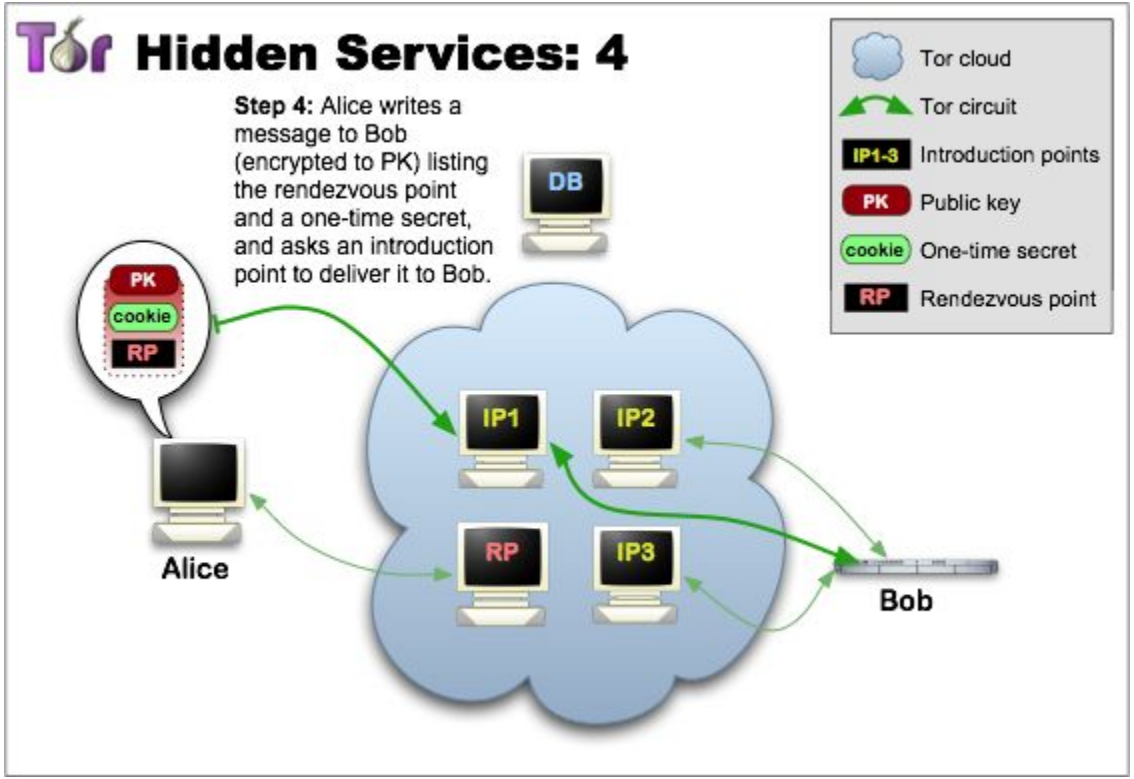
Setting up a hidden service (2)



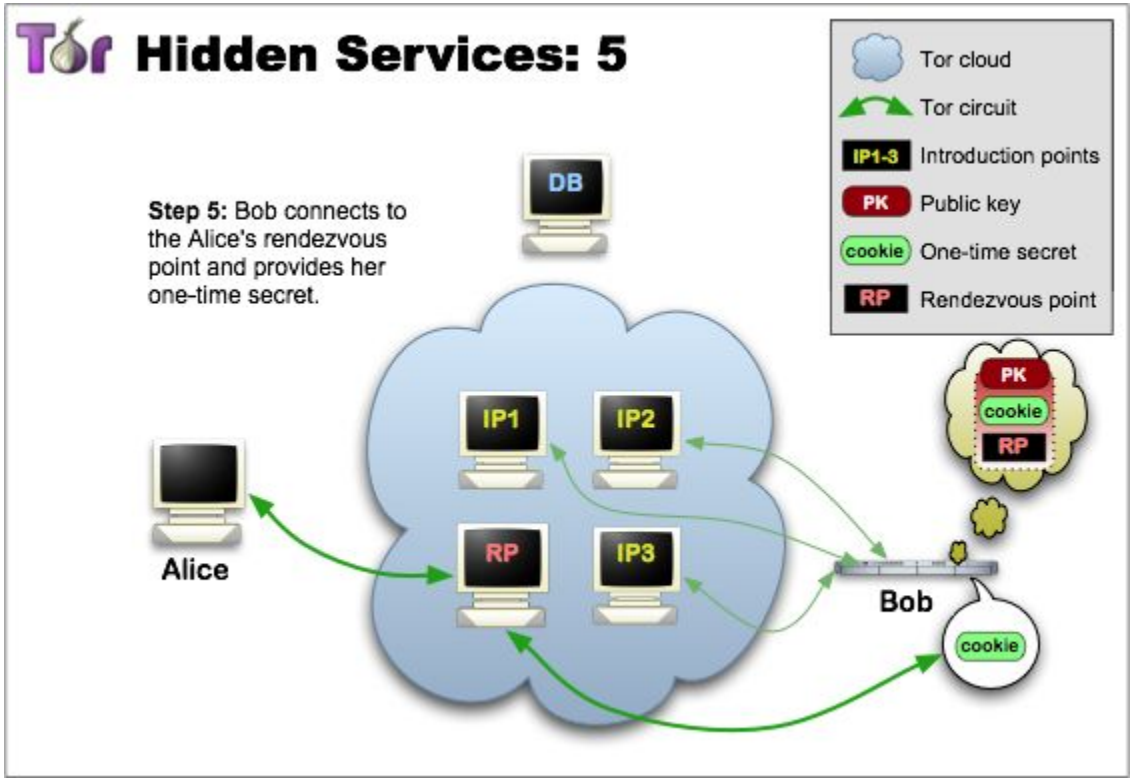
Setting up a hidden service (3)



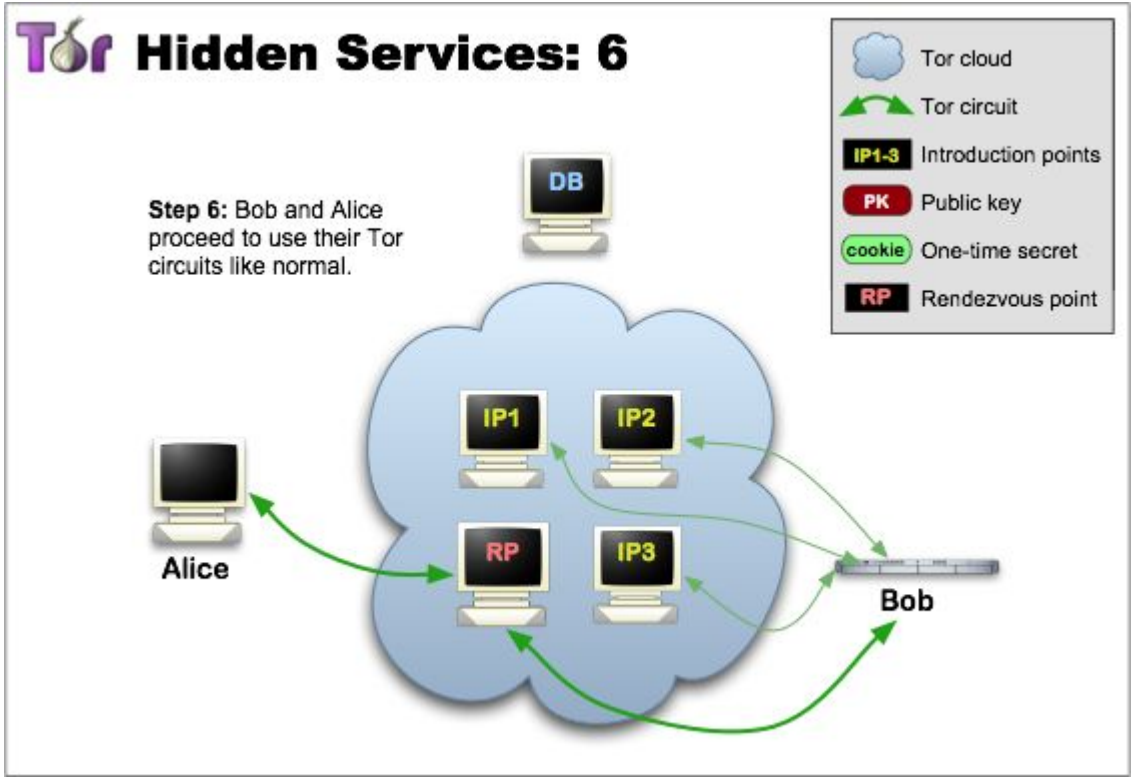
Setting up a hidden service (4)



Setting up a hidden service (5)



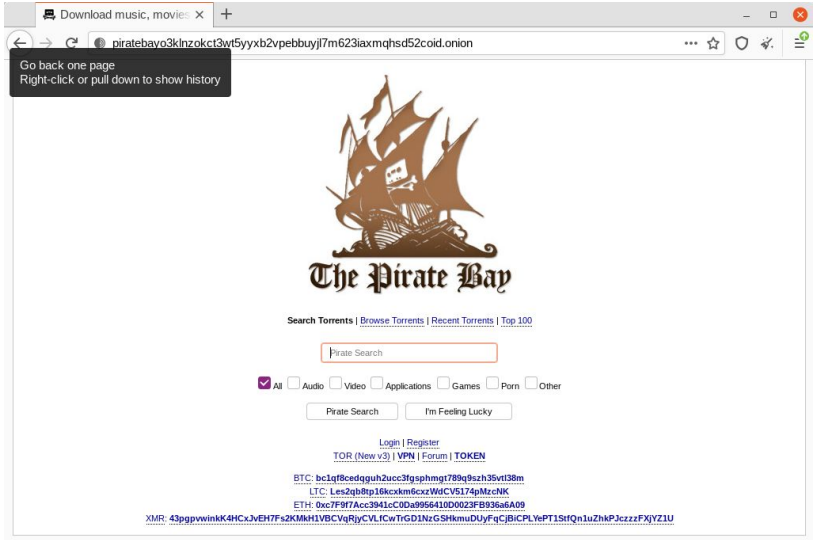
Setting up a hidden service (6)

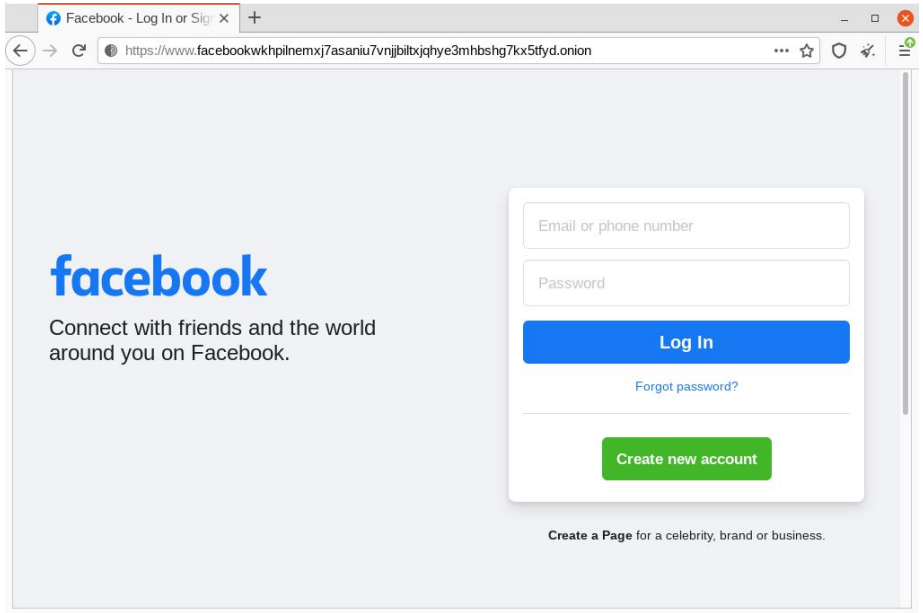


Steps

- Bob generates a long-term public key pair to identify his service
- Bob chooses some introduction points, and advertises them on the lookup service, also providing his public key
- Bob builds a circuit to each of his introduction points, and tells them to wait for requests.
- Alice learns about Bob's service out of band, retrieves the details of Bob's service from the lookup service
- 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 introduction points, and gives it a message (encrypted with Bob's public key) telling it about herself, her RP and rendezvous 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 Alice'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 the two circuits. Note that RP can't recognize Alice, Bob, or 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.

Hidden services examples





More insights

- https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf
- http://freehaven.net/anonbib/
- https://www.torproject.org/docs/hidden-services.html.en
- https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt