CS203 Gene's Class - Privacy and Anonymity Concepts

LEC9-anonymity.pdf
PDF

this is another slidec, read extensively

Thanks! This slide deck (LEC9-anonymity.pdf) is a deep dive into privacy, anonymity, and various systems and protocols designed to protect them, particularly in the context of public networks like the Internet. Here's an extensive, organized walkthrough of the key topics discussed:

---

1. Privacy vs. Anonymity

- **Privacy**: The general right to control one's personal information and to be left alone. Public perceptions are variable and shaped by modern digital life (e.g., social media).
- **Anonymity**: Specifically refers to hiding *who* is performing an action, within a crowd. It is about indistinguishability.
- **Unlinkability**: Can't connect actions to the same actor.
- **Unobservability**: Can't tell if any action took place (even more strict).

---

2. Threats to Anonymity

- **Passive traffic analysis**: Observing traffic patterns and metadata.
- **Active analysis**: Injecting or tagging packets.
- **Node compromise**: Trust is hard; assume some nodes are malicious.

---

3. Chaumian Mixes (1981)

- A trusted mix server decrypts and shuffles messages to anonymize sender-recipient links.
- **Example**: If 3 people send encrypted messages through a mix, the output is in a random order—an observer can't match input to output.
- **Downside**: Public-key crypto for every hop → high latency.

---

4. Mixnets and Onion Routing

- **Mix Cascade**: Sequence of mixes to improve security even if some are compromised.

- **Onion Routing (Reed, Syverson, Goldschlag, 1997)**: Sender encrypts routing info in layers like an onion. Each node peels one layer and forwards.
- **Each node knows only the next hop.**

---

5. Tor (The Onion Router)

- **Telescopic circuit setup**: Progressive route building.
- Uses symmetric crypto after initial key exchange.
- Thousands of volunteer nodes.
- **Supports low-latency communication**, unlike mixnets.

Tor Circuit Setup

1. Establish key with Router 1.
2. Tunnel through R1 to create key with R2.
3. Tunnel through R1, R2 to create key with R3.
4. Use circuit to communicate with destinations.

---

6. Location-Hidden Services

- Allow deploying a server whose IP is hidden.
- Uses intro and rendezvous points.
- **Silk Road** used this method.
- Resistant to censorship, DoS, and even physical takedown.

Setup

1. Server advertises intro points via directory.
2. Client uses intro point to send rendezvous info.
3. Server connects to rendezvous point.
4. Both connect anonymously.

---

7. De-anonymization (Silk Road Case Study)

- Ross Ulbricht's capture wasn't due to Tor but rather:
  - Operational security errors (fake ID, a post on Stack Overflow).
  - Weak endpoint security.

---

8. Other Anonymity Systems

- **Mixminion**: High-latency, mixnet-based email.

- **Mixmaster**: Similar to Chaum's mixes.
- **Free Haven**: Bibliography and resources on anonymity.

---

9. Dining Cryptographers Protocol (DC-net)

- **Goal**: Send a message anonymously with *information-theoretic* guarantees.
- **Three cryptographers** example:
  - Coin flips shared between neighbors.
  - Payer lies about their coins.
  - "Same" vs "Different" counts reveal if NSA or one of them paid.
- Untraceable because everyone shares noise.

Key Feature: No one except the sender knows who the sender is—even with unlimited computation.

---

10. Super-Posed Sending (Generalized DC-net)

- Works for *N* participants.
- Each contributes random bits.
- XOR trick ensures message bit appears only once, others cancel out.

If you'd like detailed walk-throughs of any particular slides (e.g., Mix design, Tor circuit setup, DC-net math), just name them and I'll expand accordingly.

this is the prof's lecture transcribed,Read it very well because I will ask in future some questions from the slide and you need to answer me from both the slide and the professor's lecture.

right there's this single trusted benevolent mix that basically well
batches and mixes all this traffic right and of course if only one person sends
an email and only one person receives right it's clear who sent and who received
if the adversary can observe all the links that are incident to the mix right
that clear but anonymity loves company the more people use the mix the better
right so popularity is good because the more traffic more people send mail then
it's very difficult to map it the more difficult becomes to map incoming
messages into outgoing messages okay even if you have like even in the worst case
like a one sender right but if that one sender sends a lot of mail to different
recipients even then mix can help because it can mix those messages and it would
be difficult to see at what time did this sender send mail to this particular
destination okay so you still get a little bit of help even if there's only one
sender but of course in the real world you want mix to be heavily used by lots of
people
right so how do you reply if you send an anonymous mail message right previous

slide really showed you how to do that right Alice knows Bob's address Alice
encrypts a message for Bob obtains Bob's public key from some directory
right here's Alice you know she knows Bob's public key she knows that Bob's
existence otherwise why would she send email right so she has a secret message to send
it and she doesn't want anybody to observe the fact that she sent a message to Bob but
that's fine for one-way communication okay for example this is a whistleblower
reporting on some abuse but what if this is a conversation what if Bob now needs to
reply this doesn't solve that problem right there's no reply here because when Bob
receives the message he doesn't know who sent it unless Alice explicitly tells him
inside the message that she did but otherwise it's just a message destined for Bob from
someone now how does Bob reply to that someone without knowing who that someone is so
same idea as before right same idea Alice sends a message to Bob for the minute but now the message that
Alice prepares has format has some structure inside that message well there's actual text whatever she wants to send
but there is a preamble that has this stuff in pink all right there is that a key
symmetric chosen by Alice at random Alice's name in this case she doesn't mind telling him
I'm sorry sorry in Alice's name but it only cryptic under the public key of a mix so
actually Bob won't find out who Alice is in this case now if you your eyesight is good you will see if it says
it says mix try because Alice for the return correspondence for return email could choose a different mix
doesn't have to be the same mix that she used on the way to Bob okay same idea but this she may choose a different
right right and then k2 okay and k2 is a fresh public key okay and sorry k1 is a fresh public key and mix mix
and he says prime is possibly different doesn't have to be the same thing
so now how does uh how does Bob reply so the first message follows that first blue
arrow second blue arrow just like in a previous slide the same thing the mix does the same thing
I mean of course I don't show you other people who send mail right so of course you assume there's other
people so but I'm just focusing on the communication between Alice and Bob so that's the message as
before what's what makes sense to Bob is uh pretty much the same thing except Bob now knows that inside
m there is that magenta part mk2 okay and so the reply message from Bob that has a different format
it doesn't have the same format okay it's not it doesn't have the same format as the original
initial message from Alice the reply message has two parts the pink part and that's taken verbatim from
inside the message and the black part which is a you know the m prime that that message that that bob wants to
reply to ask right he wants to say something back and that message doesn't need to have any structure
it's just a message i'm proud and all that is encrypted on the k2 that's a public key
so far so good
so the mix knowing that this is the return message it treats returning messages or reply messages
differently from the forward going messages right he knows that this message has two parts
so what it does it decrypts this pink part right because it's encrypted with its public key
and inside the pink part he sees a and a key k1
and what he does and he does what he's instructed to do which means he takes

k1 and takes r2 m prime the black part over there encrypts it with k1
k1 is a symmetric key k2 is a public key okay so this is a symmetric encryption
and addresses this to a so this a is visible in the clear text
make sense
so a gets it a knows k1 because she generated it originally
okay okay she will decrypt the outer encryption and then because she also generated k2 right
she knows the corresponding private key to decrypt this encryption so she will obtain him
so what is this here right this is an anonymous two-way channel between alice and bob alice knows
who bob is right somebody needs to know this in the beginning
otherwise what's the point of communication but bob does not know who alice is
just he knows it's someone but he doesn't know who
so what is this good for well you know many types of interactions right
like uh whistleblowing right it's also great for confession
you want to make an anonymous confession kind of like certain religions allow you to go to a
temple church and anonymously right go into a booth you've seen it in the movies and you have
done
yourself you go into a booth and there's a partition it's a dark little booth and you sit on a little
leg there's a partition there's a mesh you know there's a person on the other side you can see the
silhouette maybe but you don't know who they are and that person more importantly does not
know who you
are but generally they confess the person making the confession knows that it's a priest
or some kind of other religious character right because when you go to church and make a
confession
generally not who the priests are in the church but the priest does not know who you are right
ideally in the physical world
not likely they probably know because you know they can open the door after confession see who
walks out
or they can recognize the person by voice right or by what they're confessing okay so but in the
physical
world we know how this works right so this is can be done much better in this world alice says i
want to
send a confession to bob bob is a religious character person who like a cleric she confesses her sins
and
she gets enough solutions back and bob learns nothing about alice other than her sins
nope not compelling to all i'm sure there are other applications you can think about
okay any questions
fine now why would you want to rely on one mix one mix is nice because you're just wild you don't
need to
you know create any kind of big infrastructure you just dedicate a huge computing power
and storage power to this giant mix and you presume that some benevolent entity runs it and it can
never
be hacked because boy if it can be hacked you know bad things will happen but what if you didn't
want to
rely on one single point of failure because the other problem is that if the mix just like goes down
or subject to a massive denial of service attack well then the buy anonymous mail uh but more
importantly
it's difficult to find one entity one organization that will run a global mix so for that reason one
what child that same character that designed this thing and back in the early 80s suggested using
what's
called a mixed cascade or a mixed network this is basically what the word suggests right it's just
a sequence of mixes and generally operated by different entities like one operated by an electronic

frontiers foundation one by aclu one by you know uh say the whales and one by you know icelandic whale
hunters i don't know and whatever whatever whatever so mutually like suspicious entities but they're all
interested in operating at least one of those so you can imagine them sending your messages through not
one but one but several okay one immediate benefit is that that first mix is compromised or controlled
by the adversary you're still okay because the second mix is now in fact even if both here the first and
the last are controlled by the adversary you're still okay because the middle mix is not controlled by the
others and because mail is by its very nature not a real time not a synchronous protocol right it's
what's called stored and forward in other words asynchronous meaning that you do not expect mail to arrive
the second you sent it yes yes we do sometimes we do when we wait for those second factor
one-time code notifications right we expect email to arrive very quickly and sometimes it doesn't
because it gives you no guarantees okay text messages that's another story right what's up text etc
they usually try to make things very fast and they don't guarantee but you assume that things unless
you are disconnected from the internet or cellular network you expect uh like these kind of instant
messages right with sms apple i message what's up signal telegram etc they arrive quick but email
makes no guarantees because it is stored and forward yes so if that one mix in the middle is still honest
well you can mix enough messages to cause confusion essentially prevent the compromise or the adversarially
control basis from learning who is sending messages to whom right in other words it will try to foil
this correlation attack which incoming message corresponds to which outgoing message
but as you can imagine
there's a lot of overhead here okay because public key encryption especially like the layer because
what we need to do in a mixed cascade you solve it you do it with one mix right this but if you have
multiple mixes you see those encryptions that have that alice and charlie and david perform here
here well if you hop through three mixes then each message needs to be encrypted three times
which is expensive it's expensive for senders but it's also expensive for mixes now they have to
that they have to decrypt
and so it's expensive and high latency is encouraged because well as i said messages are not instantaneous but
still even for something like email latency can be high because every mix on the path to the destination
needs to batch enough incoming messages to mix them right if it's if a mix receives and sends immediately
that's not very good right because anyone eavesdropping on that mix will see a correlation between incoming and outgoing message
so that's the whole point is that there isn't mixing right now there are other things you can do
i mentioned earlier that you can uh generate spurious traffic meaning chaff right garbage
you confuse the needs dropping out your certain stuff that looks like it's an encrypted message
but in fact when they make when they're receiving mix decrypt it you realize the garbage throws away
so every mix could always generate a lot of that kind of chaff but that's expensive too
and generating is not but receiving it is expensive because you have to decrypt try to decrypt it
in order to realize it's got it so in uh today's world uh mix nets are still useful

somewhat and there are um actually operational mix nets out there that you can use
uh even for free but they're not very popular okay most people today want anonymity for other things
and primarily they want anonymity for web browsing okay not so much from watching movies but because they
are small but even for watching movies sometimes right accessing let's say uh well everybody's favorite
of course is porn but then there is like streaming services that uh are not always accessible from
where you are or being on your local provider does not allow you to connect to some let's say streaming
services so people often want to use an anonymous communication for that purpose okay
so if you're a whistleblower or something like that email is fine right if you want to send an anonymous
denunciation message and complain about something you know it's fine to use anonymous mail but if you want
to browse the web if you want to use uh a remote terminal right like a remote login you can't use makes that
too slow right so today that the preeminent uh means of anonymous communication on the internet is torn
the onion browser and uh basically it works like this so pretty much all anonymous communication methods for
low latency or synchronous traffic what that is they use public key cryptography kind of like makes it but
only in the beginning and so they use it to establish what's called a circuit the virtual circuit
for a sequence of anonymous routers for a sequence of anonymous routers and then once the circuit is
established the traffic the bulk traffic flows through that circuit but no longer using uh public
key control it then uses symmetric which is much cheaper make sense all right so you pay a cost to start
anybody here ever use a tor browser or any tor application yeah so you know that when you start it
there's a delay right why is there a delay exactly because your core browser is probably as
any of you modify any configurations on the program so you just use the default
okay use the default but the default is i think three pops right if i remember correctly
one well i'll tell you more about it but essentially it has to pick three routers from some directory
which we'll talk about later and it has to establish these public key based initially circuits through
those three routers and once you establish a circuit it will tell you on your browser right
connection established right remember that when you start to store browser it says connection established
then when you when you type in a website or click on something it's reasonably fast right it's not terrible
not terrible slower than your normal graph for sure but it's not terrible
okay and that's another connection and it doesn't matter how many websites you go to
you're probably using the same circuit unless you specifically call for like there's an option say
new tor circuit for this website and if you just use a default no matter how many websites you go to
it goes for one circuit okay and so at every hop in that circuit there is encryption there's decryption and
re-encryption decryption and re-encryption why because just like in the next example if anybody is
listening eavesdropping on our on an anonymous router sees what comes in what comes out they should not be
able to correlate incoming to outgoing okay but if you listen to be careful you might be wondering how

is it that asynchronous communication right how do we can't delay it right so he comes in and out comes
in and out you must must do so right cannot be sitting there waiting for enough other traffic right so that's that's that if you're thinking about that that would be an interesting issue an interesting question so the way that these routers work is they also rely on many simultaneous connections that is many users using every single router and if many users are using every single router or synchronous communication browsing the web we won't log in whatever then there's always packets coming
into that router so in other words it does not retard them or mix them in any way it tries to process it as fast as possible but it but the way that it achieves a lack of correlation between incoming and outgoing
traffic is by expecting to receive a lot of stuff okay
so the design of uh under routing goes back to mid to late 70s 90s and there's a it's kind of a seminal
paper called uh by under routers by beat cyberson galschlag and uh basically it's it's what you expect
you have a set of routers represented by circles here some of those routers are malicious untrustworthy okay and some are transporting but you don't know which ones okay so you as a some as as alice a user who wants to use a design of routing you just pick three
out of the set some directory of anonymous routers in this case r1 r2 r4 sorry r1 and 2 r3 r4 sorry four hops here and uh the pink ones of these r's are just regular routers you didn't pick the red one the red circles represent the ones that are uh controlled by the address so you also it so happened alice obviously did not know this but it so happens that r3 is malicious so she sets up her circuit and she can't know which ones are malicious which ones are not all she expects is that at least one is not malicious
okay of course alice controls the length she could say you know i just want to go for one router it it would be stupid kind of not recommended or she can say i want to go for 10. that's all stuff done you could configure uh core to get on any other one in routing you need to go for as many routers as you want the more you go through the slower your connection will be clearly because only routing is not routing in the internet sense of the world remember we talked about bgb and like internet routing
nobody has any recollection there was a full system reset since then
new version of software was installed no uh remember routing for ip right when i say routing here i don't
mean that that that onion routing works at the application level so you install like a software package and it does this kind of routing it's essentially what's called an overlay right everybody heard that word before it's an overlay so it runs way above id okay and one of the reasons it does so is
because if you want community participation right they strengthen numbers the more onion runners that there
the better you don't need to have good people just to run an application program make sense so you
can install tor and on let's say you don't want to start in your laptop that would be really foolish for many reasons but if you have a desktop machine not in student house they'll they'll crack down on you
like immediately uh they monitor that stuff but if you have i don't know in your home off campus or your
parents home or whatever you have a desktop machine or at least a laptop that's pretty powerful you can
install tor yourself and become a tor relay or tor router not difficult right clearly you'll be sacrificing something right so your your your device will slow down not right away because actually tor routers

are
not trusted right away because if you're a newbie nobody trusts you very little by little it's reputation based anyway back to our regular schedule broadcast so you control the power and uh that's pretty much
it how onion routing works but why is it called an onion routing is it because it's smelly spicy edible any idea
layers layers of encryption layers
because if you go through four routers here and alice has to hop for one one or two or three or four four she better encrypt whatever message she sends first in our fourth public key then in our fourth key
then on top of that r3 key r2 key r1 key okay then she sends message to r1 r1 peels off the outermost layer
you see why onion then okay good
better illustration
now this is around establishment but this is uh hypothetical only this is with public key this is not how four works this is how it works it it's a good illustration of uh uh on your right so here is a message
that alice wants to send to bob that message is encrypted under bob's public key
why well because eventually that message should reach r4 and r4 is what's called exit node right exit the network the overlay network and so if you don't encrypt it then the message will be in clear text right arriving to bob sometimes may be okay but generally not a good idea so next layer see here
k4 that message on the public key is encrypted with k4 what is k4 the message sorry the key that alice shares with r4
the last part what is this here this part that right is encrypted on the public key of our board
okay okay and it contains k4 okay and it contains k4 so this is a little bit this is like a header to do a little header and this is the actual body so let's follow again then all that that up blob gets encrypted
out with k3 but k3 itself with r4 is here okay and so on and so on so it's one two
three four five layers of encryption
okay so let's remember animation here yeah okay so let's consider this to be the big onion big fat onion
five layers of encryption
alice sends it to r1 r1 removes that layer you see why because r1 sees this
he decrypt this he decrypt this with his private key learns r2 learns k1 okay now he's gonna decrypt that k1 get rid of this encryption
okay and send it to r2
bam
okay okay r2 decrypts this with his private key finds r3 and k2 and now decrypts the rest of the message
with k2 so the onion gets smaller bam okay r3
r3 well same thing right rinse repeat bam r4 and bob receives the innermost message
now yeah the r3 is still adversarial right it's still an adversary controlling it but what does the adversary learn
when r2 who is honest processes the message sends it to r3 r3 receives it and sees what he might know
who forwarded it he probably does it's r2 forwarded it he knows they received it from r2 and he knows the
next top is r4 does he know that r4 is the last top you know
r2 maybe maybe not so all he learns all the adversary learns is that okay uh i received it from r2 and i'm sending it to r4

so
right you see that still doesn't learn anything useful now if it was r1 that was uh adversarial
right if r1 was malicious instead of r3 then r1 would learn that alice is sending a message to somebody and the next top is r2 but that's it
the adversary could control r1 of course would know that he's an m3 node the first top
if the adversary controls controls r2 instead of r3 exactly the same situation as r3
knows the previous top does the next top does not know if he's first or last
if the adversary controls r4 he knows he's last he knows he's last because the next destination is not an onion runner okay
but he doesn't know how many hops before and he doesn't know where it comes from of course
so tor today doesn't quite work like that that was how it used to be
uh it's what's called second generation
on the routing network uh first generation used what we saw before
uh this one uses what's called a telescopic round setup okay uh i'll explain what that means uh if you
want to read about it download stuff uh from uh related to tor it has uh plugins it has uh
uh the the the self-standing tor browser that there are apps for um ios and android uh all kinds of uh
notebook so there's plenty of platforms supported by tor okay so if you want to use it doesn't matter
what device you can always install um it is designed for anonymous low latency communication and it's
been running since 2003 it's quite a long time and there have been many attacks but there have been few
successful actually no no real successful attacks on tor this is not that i've heard of and that would have been probably publicized well if there had been um there are thousands of tor nodes like maybe
10 000 tor nodes worldwide there are hundreds on every continent uh there are as of three years ago
between three to four million regular users now regular users are people who use for like daily basis
there are surveys done um but there are probably 10 times as many occasional users
um it is easy to use actually i don't know i wouldn't even use quotes it is reasonably easy to use if you just take the default if you want to dig into the guts and configure it yourself you could also shoot
yourself in the foot by doing something silly or you could slow it you could slow yourself down tremendously so
most people just leave it deep there are situations when
uh you go to certain countries that uh don't like anonymous communication and they will block tor
okay and so in those countries you need to use what's called door bridges
which is additional like very temporary relays for relays that are too short-lived to be blacklisted
by the authorities so i've been in that situation myself in countries where
let's say freedom is a little tough to come by and then the tool works right you just need to use
what's called bridges and it slows it down all right so the proxies plugins it's free it's great
uh the only piece of software that i will advertise in this class if you are ever want to do anything
i don't understand how does it work well it it is still kind of an onion network
and it imagines all these uh multitude of uh tor nodes right onto these properties now because it's an
overlay right it does not correspond to geography or internet connectivity you see what i'm saying
what overlay means is that overlay is essentially an any to any network that's why you see lines drawn
into a different light so these are virtual connections virtual lines these are not physical

lines obviously right it's not like this node is connected to this node there could be like they could be a different content but logically they can communicate directly because of the application of layer so the client alice who uses tor
in order in order to set up a circuit she picks today three nodes the default so she picks the first entry node and she establishes a connection and that connection is established because you public it
okay public it because she well i'll show you later but there's a directory of tor nodes that is the download from the distribution from the software distribution side as a directory and you select the tornadoes the three tornadoes you want to use randomly from that directory
and so one of them will be the first and it could be very far away from you or it could be very close doesn't really matter so you establish a tor connection right a direct connection between here and there
so think of it as a kind of an asset if it helps you think about it as like a tls on ip set connection like a pipe secure pipe but it's not that the network layer or or or session it's at the application okay so anybody who observes this right who is dropping on alice here will know that she's connecting to some tor
or no no no right somewhere but they cannot see the traffic now the second hop right the second router
alice is going to use the already established connection the the red line as a kind of a secure pipe for which to insert an extension to the second round that's why it's called
telescopic telescopic like like a you know like an antenna that you stretch out an old-fashioned old-fashioned radio tv kind of like that or old-fashioned pointer that you start position
so she basically does this so the second
one
router two i think i'm missing the router two here
oops so anyway the point is i'll just get so she uses
the green key you see protects well the green key corresponds to the security of this connection here okay the magenta key is the second connection that is established and through this now to this second half and then the green key sorry this mustard the brown color key is the one that is used to establish through here to this so they are like one inside the other right the only the first connection is established directly from alice all the others are established through other nodes and then you see
those red things going to two different places and why because once a tor circuit is established you can use it for many sessions like on the web it doesn't it doesn't be used just on the web but mainly
it's used for the web so if you're using tor browser you could have two different tabs or two different windows and one could be going on to cnn.com and one could be going to whitehouse.gov it's the same tor
circuit right that's carrying traffic from all and it's only demultiplexed here
and the last hop or exit does that make sense
the telescopic nature of the route is attached
no
yes i'm sure
but once the route is established that packets are encrypted and re-encrypted additionally right so alice encrypts here it's decrypted re-encrypted so in here decrypted re-encrypted here decrypted re-encrypted
boom destination
so what does the adversary see if the adversary is there while yoni i can say before sees that alice is talking to a tor node right so uh the adversary might be able to decide if the adversary looks at the ip address of the of that node right that yellow node with which that is uh the first hop because the adversary can also download tor and look at the directory and say ah this is a tor node

okay yeah
yeah there's denial of services i mean tor nodes can come down i mean it's a power object
yeah you just reestablish a server for a different set of nodes
but you don't know that the message didn't get delivered
you have no idea but it doesn't matter because i mean this is your graph like if you're browsing the web it just the pre-establish connection wait a few more seconds be all right i mean yes there is a tor node can always be performed analysis yeah it can drop back well in that case your connection will be unstable you can just always click on like new tor circuit
if the adversary is here all you learn is that it's in the middle of the tor connection so between two tor routers and uh that's all so somebody is using tor but there's no idea who or how many people
are here same thing here is the worst part right because here if especially if alice does not include it right for bob this would be a problem because that would actually reveal that it's out right
like if there's some kind of a web form right the post an http post where she fills in the name and address right so obviously if you're using like tor browser you should be using only https
right right right or tls ssl right
well you can use http but then you don't enter any information okay you have to make sure not to enter
any information
so as i said before many applications can share one circuit uh that's also true but you know if you're using different applications not just web browsing then it's a little difficult right so maybe on the on the phone that's more possible uh tor router as i said doesn't have root privileges so any one of you
any one of us can install a tor router and register it in uh in a directory there's a slow reputation building process you won't be used right away but after a while you know if you if you basically reputation is based on like where you are what your speed is like performance and also like uh what is your hours of operation like are you reliable you 24 hours are you only available at night so on some some routers have restrictions on timing uh directory service so this is
a bit of an issue right so you need to know who the tor routers are
and um today there is this foundation right the electronic frontiers foundation which is a
a non-profit organization that essentially kind of like it's a little bit like aclu but it's not a
it's not a legal defense fund i mean like aclu it's really just about privacy like supposed to promote the right to privacy right so it's all about it's not only about privacy but if you go to efr.org you can read all on your own uh they also distribute torn right so the door is mainly distributed through that website it's like if you want to get chrome you go to google right if you want to get tor you go here um you want to get firefox you go to mozilla right you don't go somewhere you know to some other
place so it is one place so if somebody mounts an attack against dff that would be a problem
um directory servers right identities and and keys public keys of the of the of the routers and uh and sorry let me let me back up sorry the routers are identified by some ids and ip addresses and
the way you found out about find out about short routers is from what's called port directory servers and these are special nodes that its only purpose is to provide the directory of tor routers
now why does the ff not do it because the population changes it's too dynamic right so the directory servers are supposed to maintain up-to-date lists of like reputation ranked uh tor routers there are not
that many there are some i guess thousands of these directories but here's the pov the point of vulnerability if somebody hacks directory servers and compromising those keys it's possible to poison
the directory and distribute identities of tor routers that are malicious so if some let's say state level actor decides to mount us for a civil attack and creates hundreds of malicious tor routers and introduces

them into a compromised directory tor will fold and will be a major problem it hasn't happened yet
but it could happen and that's it that's the main uh issue with tor
now tor also allows you for something very nifty or very dangerous depending on the point of view
you anybody heard of this one person used it or just heard of it it's okay listen i won't i won't i won't tell
uh so location hidden services you'll see why they're locking in a minute um so you wanted to you want to
deploy a service on the internet right without a which offers a service of some sort maybe it's a digital
service maybe it's a conversation service maybe it's a service that sells physical goods like analog
but you don't want anybody to know who you are and you don't want anybody to know where it is
it could be that this is an information service for example you live in an oppressive society and you
want to publish something as a service that is uh like a subversive in its nature goes against the
regime and of course the regime wants to silence you and censor you but if they cannot know if they
cannot know where you are they cannot censor you right they can't find you they can't do anything
so it's not just for selling stuff all right it's also for publishing stuff
so there was a infamous marketplace called the silk road anybody remember it no
it was an example a mega example of a of a hidden service on on tour
so the idea about not silver but in general hidden services they should be accessible from anywhere
resistant to censorship survivable against like even a full-blown denial of service and resistance to
the physical attacks because if you don't know where it is you get physically damaging or harm
so silk road was uh yeah this is actually a uh a screenshot
this is some years ago but yeah uh you see lots of good stuff beautiful good
draft paraphernalia electronics probably stolen uh apparel computer equipment likely stolen too
psychedelic stimulants opioids ecstasy cannabis cannabis not exactly like they
drugs lots of different drugs right so uh yeah good stuff all right even pictures
so how do you create a location hidden server right so how do you how do you make this happen
if you want something like this you want an enterprising person uh so you let's say your name is bob and
you want to serve set this up so you use uh tor right to create these kind of uh uh let's call them
semi-permanent circuits tor circuits so you need to have a stable connection to the internet right
that's that's that's important and you need to set up these by the way tor does not
uh i don't believe it dictates how long the connections can last so you can keep the connection
up it stays right so you set up in this case three different connections to one tor service
three different tor circuits to three different places okay called introduction points and these are
themselves uh services they're computers somewhere they could also be tor routers themselves
okay but in general that they don't have to right so these are just like some computers somewhere
that grants them software that will quite fresh so you set up these semi-permanent software
step one okay then
step one okay you're not seeing what i want you to see
yeah what you're not seeing here is that now once these once these things are set up the server bob
gives these descriptors what's called that they're called dot onion addresses have anybody ever seen
dot onion addresses that's hidden service if you ever encountered a url that has a dot onion that's a tor
hidden server uh so this is done in the beginning only right so the server the server bob gives to this
lookup service lookup directory which is a well-known detective place on tor uh the description right points for
these introduction so they're essentially a bunch of x digits with dot onion so i have a random looking

addresses with dot onion at the end and they are published in this directory which you don't see
then you also don't see i don't know why because it looks fine on my laptop there's client alice here in that
white box and alice she looks at this this public directory of our service descriptor and says oh
oh ecstasy cheap i'm going to buy some ecstasy i'm going to rain yo okay so she gets the address of those
interpoids that's one of those dot onion addresses
okay what you don't see is she also ah this is maddening she sets up another circuit to something
called a rendezvous point which you don't see don't see here okay that's point four there is a she picks
just like just like bob picked introduction point she picks some other place on the tor network and she
calls it a rendezvous point rendezvous and sets up a four circuit there okay so this is in real time
right she looked up the service directory fished out an introduction point address for bob's ecstasy
selling service and then she creates the four circuit to a day rendezvous point which you do not see
that's step four step five once that rendezvous point circuit is set up she sends the
address of the address of the rendezvous point okay to one of bob's introduction points
you following me don't look so much at the picture just try to because it's not it's missing some of the
graphics that it's supposed to show
okay so far so good now the introduction point
informs bob yo you have a customer potential customer bob wake up bob says i don't work in these hours so
he might say no way not interested uh or if he's interested to perform the service he
he basically connects to the rendezvous point which you don't see that's the end of the blue arrow there
should be a rendezvous point there just imagine but you see what's happening now so the rendezvous
point which you don't see matches bob's incoming that blue circuit with alice's
circuit that she previously established again you don't see it right and says oh this
thing that somebody she doesn't know the rendezvous point doesn't know it's alice this thing from the
buyer or from the user matches this thing from the service and she you know connects the two okay
so now you have essentially two tour circuits one from serve above to the rendezvous point and one from
client alice to the rendezvous point make sense again don't believe the lying guys don't look at that picture
so you have a concatenation of two tour circuits managed by that random point
and now the transaction can proceed okay
and this is the ross umbrick that was the guy i mean i guess he moved then he doesn't look so happy today
he spent time in federal prison until our president recently pardoned him
but yeah that guy ross umbrick was the operator of the of the uh so-called marketplace and was arrested
from 2013 and i think a couple of years later was convinced convicted and sent to federal jail
um it's a nifty operation you you could also what you didn't see there you could also buy murder for
hire yeah you can get somebody's like busted kneecaps puncture tires anything lighter than murder too
but murders are fine uh nobody knows how many were bought and sold through that uh so throughout the site
um one of the interesting uh uh posters there is that nobody knows really how it was how he was

trapped right because it's a it's a hidden service right so it wasn't torn at least most people in the security community believe that it was not a weakness of tor that the federal government despite all of its
power could not penetrate tor partly because they have no jurisdiction over torn notes overseas outside the
united states right no jurisdiction whatsoever uh so there are three speakers only the last one makes
sense makes more sense there was a package of peccadees from canada that was traced to an apartment in
san francisco where he lived that was one theory there was a fake uh murder for hire that was arranged by
law enforcement and apparently trapped him again no no proof for that more likely the last one
familiar with stack overflow right so somebody accidentally under the name olbrick ross olbrick posted a
question
before like way before how can i connect to a torn hidden services in curling php and then a few seconds later they changed the name because realized he'll you know revealed his real name change the name to
frosty too late right stuck uh you know stack overflow like elephants never forgets so somebody took a
snapshot of that oh yeah and uh this last bit helped the encryption key of the server ends with substrate frosty a frosty so that's another catch right uh tor is not the only thing out there but by far the most popular there are others uh pre-handed has an excellent biography biography that you can read if
you're interested in the field mixed minion mix master still exists uh but they're not very popular i mean you can use them
there they're they're mostly uh they're quite slow right but they're different so slightly different interpretations of the challenge mixed cascade okay now let's see quickly i'm going to show you something
that is a bit out in the left field uh there's pure cryptography or at least on the surface so the idea here is to show you that how to achieve the best anonymity possible like not relying on anything else
and the idea is a little crazy and it's also due to the same person ciao that crazy guy from the 80s and it's called dining cryptography problem okay so but the idea behind it is to do this how to send a
message in a message in a group without right without any kind of transibility you want to reveal a message send a message in a group let's say like our group here but nobody knows who sent the message
okay imagine that so this paper from 1988 shows how to do it and they they kind of really interesting
part about it is what's called information theoretic anonymity it gives you this information theoretic anonymity meaning that it's not based on any mathematical problem it's not based on any kind
of assumptions okay that's uh question
that's the most beautiful things it's difficult to make practical so here's the scenario three cryptographers we're gonna start with three because that's the easiest part right but it's generalizable
to a groupie like as big as ours bigger or any of any size three cryptographers are having dinner okay in a restaurant now either nsa national security agency is paying for dinner or one of them is paying for dinner but if one of them is paying for dinner he wishes to remain anonymous he wants everybody

to know like one of them is paying for dinner but he doesn't know want the other two to know who that
is and he will deny of course okay let's assume they're all he just for ease of explanation so how does the particle work so imagine three people sitting around the table three people around the table
finish dinner check comes somebody paid for the check so now they're trying to figure out who paid and a safe on one of them so each person flips a coin last time i checked most coins had two sides and scales and shows it to its neighbor on the left so everybody in a circular table has a neighbor right and left you flip a coin show it to your left-hand neighbor everybody does the same thing okay okay every diner right every cryptographer sees two points their own right and the right neighbors
so if i'm one of those cryptographers i know my coin flip and the guy over here you're going to show me
his that's the problem each person then publicly announces whether the two coins are the same both heads or both tails doesn't say which just says if they're the same like for example if i throw heads and the person on my right shows me tails i'll say different if i throw heads and that person shows me heads also i'll say same so here's the rule if the number now the all three all three of them say the same say say something right each one must say either same or different that's the only two words allowed if the number of same is one or three then nsa is paid no gathering
if the number of same is zero or two then one of them paid and nobody knows who
main thing is non-payer the person who did not pay does not know which one of the other two it is he knows one of them paid the non-payer knows he didn't pay right
but he doesn't know which one of the other two paid yeah what's your question so if the parity of the guests changes instead of the parity of the answers let's see it's coming right
so here we have pictures one thousand words three cryptographers first and on the side you have going to say who may have paid for dinner check all right somebody paid right think about what if you are not a payer you're one of the critographers who did not pay you know you didn't pay and you see
the same point right meaning that meaning that you you basically threw heads that was your point and the guy over there showed your head right and that's so what's what's what's like one possibility
is the guy over there declared same and the guy over there declared different right that's possible what could you say about this picture because remember this is yours right you are this guy
this is your point you showed it to this person he said same
what must the other one must have been heads right
but if he threw heads
if he threw heads and he showed the guy over there heads
then that guy and that guy showed you him then that guy also should have said same
somebody's lying
the payer lies right
but who is the payer are you following who is the payer right because this is obviously the case that one of
them paid
right again somebody showed you this you threw this means you have the view of the same you said same
the guy on your on your left said same which means that if he's not lying he saw
heads and he threw heads and he showed heads to the guy on his on his left which says different but wait he showed
you this what the hell but you can't say that that guy is lying because it would be this guy who is lying
they all look generic

could be this right this could be a pair or could be this equally likely
believe it right
okay okay now if you are not a payer but you see different coins you threw tails and you saw heads so you announced
different and the other people said same same
well
you showed him tails you showed him tails
he showed head which means he is the payer if he's lying potentially or you showed him tails
and he did have tails he showed the guy on his left tails but he is the payer who lied equally likely
so you see in this case we're coming back to this rule
that if the number of same is zero or two then one of them is paying because one of them lied
in this case the number of same is two
in this case
different right the number of same is two
now the idea is that this is a generalization something called super post setting never might
this slide is too much but the idea is that you can send essentially a message that i paid
in a group like it's a one-bit message that says i paid and nobody knows who you are
so the same generalizes to a group of like our size or five or six cryptographers right it'll be the same
okay
just to extend the protocol a little bit but the idea is that if one of them paid they will announce the fact that they paid but nobody knows who paid and that's the best kind of anonymity you can achieve
all right
let's see
yeah let's let's finish here
we only have a few minutes no point in getting into the next exciting topic which is spam

Got it. I've read and fully internalized both the lecture slides and the professor's detailed explanations for CS203 Lecture 9 on Anonymity. Feel free to ask questions anytime—I'll make sure to answer using both the slide content and the professor's commentary.

explain
Encryption hides payload, not routing information
• Even IP-level encryption (tunnel-mode IPsec/ESP) reveals IP addresses
of IPsec gateways

This statement, from slide 3 of the anonymity lecture, makes an important point about the **limitations of encryption in preserving anonymity**:

🔒 "Encryption hides payload, not routing information"

When you use encryption protocols like **TLS (Transport Layer Security)** or **IPsec**, they **only encrypt the data portion (payload)** of a packet—not the metadata or routing information in the headers.

⚠️ This means:

- Anyone observing your network traffic (e.g., routers, ISPs, adversaries) can **still see who you are communicating with**.
- Specifically, they can view:
  - **Source and destination IP addresses**
  - **Timing and volume of packets**
  - **Packet sizes**

So even if your **emails, web traffic, or VPN tunnel content is encrypted**, it's still clear **who is talking to whom and when**.

🛡️ "Even IP-level encryption (tunnel-mode IPsec/ESP) reveals IP addresses of IPsec gateways"

Tunnel-mode IPsec with ESP (Encapsulating Security Payload) encrypts **entire original IP packets**, including both header and payload, and wraps them in a **new outer IP packet** for routing.

However:

- The **outer IP header**—which is needed for routing—**still contains the IP addresses of the IPsec gateways**.
- Therefore, a passive observer can still **see that two endpoints (e.g., two VPN gateways) are communicating**.

🧠 **Professor's Clarification:**

The professor emphasized that:

- **Encryption ensures confidentiality**, but **not anonymity**.
- Encryption **hides what you're saying**, but **not who you're saying it to**.
- Even with **tunnel-mode IPsec**, you only hide the identity of internal hosts; **the gateways' IPs remain visible**, which allows adversaries to infer the existence of a connection and possibly its participants (e.g., if the IPsec gateway belongs to a known organization).

✅ Key Takeaway:

> **Encryption ≠ Anonymity.** You need additional mechanisms (e.g., Tor, mixnets) to conceal metadata like source/destination addresses if you want true anonymity.

Can you explain a little bit what is this tunnel mode and what is this ESP? I mean, I read IPSec but I somewhat forgot.

Absolutely — here's a concise and clear refresher on **Tunnel Mode** and **ESP (Encapsulating Security Payload)** in **IPsec**, drawing from both your past IPsec lecture and what's relevant to anonymity:

🔧 What is IPsec?

IPsec (Internet Protocol Security) is a suite of protocols used to **secure IP communications** by:

- Authenticating the source of IP packets
- Ensuring data integrity
- Optionally encrypting the data

It works at the **network layer**, meaning it secures data regardless of what applications are using it.

🛣️ What is Tunnel Mode?

IPsec can operate in **two modes**:

Mode	Description
Transport	Encrypts only the payload of the original IP packet. The IP header remains visible.
Tunnel	Encrypts the **entire original IP packet (header + payload)** and encapsulates it inside a new IP packet.

🧠 In Tunnel Mode:

- Used primarily **between gateways** (e.g., routers, VPN endpoints).
- The original IP packet is **hidden inside** a new outer packet.
- The outer packet has new IP headers (from the sender and receiver gateways).

  It's like putting a letter inside another envelope with new addresses written on it.

🧪 What is ESP (Encapsulating Security Payload)?

ESP is one of the two main IPsec protocols:

- The other is **AH (Authentication Header)**.

Protocol	What it provides	Can encrypt payload?
ESP	Confidentiality (via encryption), integrity, authentication	✅ Yes
AH	Only authentication and integrity of the header + payload	❌ No

ESP in Tunnel Mode:

- Encrypts the **entire original IP packet**.
- Adds a **new IP header** to route the packet between gateways.
- The outer IP header is **not encrypted**, because routers need it to forward the packet.

👁️ Why It Matters for Anonymity

> Even if you use **ESP in Tunnel Mode**, **an adversary can still see the IP addresses of the IPsec gateways** (outer packet). This reveals **who is communicating**, even if they can't see **what is being said**.

So from an anonymity standpoint:

- **Encryption protects content** ✅
- **But not metadata (e.g., IP addresses of gateways)** ❌

🧠 Summary

- **Tunnel Mode**: Encrypts the whole IP packet and wraps it in a new IP header.
- **ESP**: Provides encryption and authentication.
- **Even with ESP in Tunnel Mode**, anonymity isn't guaranteed, because the outer headers (needed for routing) still expose endpoint IP addresses.

Let me know if you want a visual or example flow!

esp vs ah? write 1/2 diff in a para

ESP (Encapsulating Security Payload) and **AH (Authentication Header)** are both IPsec protocols, but they serve slightly different purposes. **ESP** provides **confidentiality** by encrypting the payload and also offers **optional authentication and integrity**. In contrast, **AH** does **not encrypt** anything—it only provides **authentication and integrity** for the entire packet, including parts of the IP header. So, if you need to hide the data you're sending, you must use **ESP**, while **AH** is useful when you want to ensure that a packet hasn't been tampered with but don't need encryption.

explain Chaum's Mix in detail

Certainly! Let's dive into **Chaum's Mix**, one of the **foundational concepts in anonymous communication**, both from the **lecture slides (slide 9–11)** and the **professor's detailed discussion**.

🔐 What is Chaum's Mix?

Chaum's Mix is a cryptographic protocol proposed by **David Chaum in 1981** to enable **untraceable email communication** on an untrusted network.

📌 Core Idea:

> A **trusted intermediary** (called a *Mix*) receives encrypted messages from multiple users, **shuffles** and **forwards** them in a way that makes it **difficult to link** any incoming message to

any outgoing one.

🏗️ Basic Components

- **Senders**: Users who want to send messages anonymously.
- **Recipients**: The intended destination of each message.
- **Mix**: A server that receives encrypted messages, **batches**, **decrypts**, and **reorders** them before forwarding.

🔐 Message Flow (One-Way Communication)

Let's say Alice wants to send a secret message to Bob:

1. **Encryption Layering**:
   Alice encrypts her message `M` for Bob, and then **wraps** it in layers:

   ```scss
   {r1, {r0, M}pk(B), B}pk(Mix)
   ```

   Where:

   - `M`: message for Bob.
   - `pk(B)`: Bob's public key.
   - `pk(Mix)`: Mix's public key.
   - `r0`, `r1`: random padding/nonces to prevent replay or correlation.
   - `B`: Bob's address.
2. **Send to Mix**:
   Alice sends this encrypted message to the Mix.
3. **Mix Operation**:
   The Mix collects a **batch** of such messages from various users (e.g., Alice, Charlie, David):
   - Decrypts the outer layer using its private key.
   - Extracts the inner ciphertext destined for Bob.
   - **Reorders the messages randomly.**
   - Forwards each to its final destination (e.g., Bob, Eve).
4. **Result**:
   Even if an adversary watches **all incoming and outgoing traffic**, they cannot correlate which sender corresponds to which recipient due to the **random ordering** and **encryption**.

   > 🔁 This is **batch processing**—the Mix waits until it receives multiple messages before forwarding them.

🧠 Why Does This Work?

- **Encryption**: Prevents intermediate nodes from seeing the message contents or recipient.
- **Shuffling**: Breaks timing and ordering correlations.
- **Padding (r-values)**: Prevents ciphertext matching or length analysis.

🧭 Professor's Insights

- **"Anonymity loves company"**: The more users send messages through the Mix, the better it works. If only one person sends and receives, anonymity is broken.
- Even if **one sender** sends multiple messages to **different recipients**, the Mix can still obscure **which message went to whom** (to some degree).
- The Mix **needs to be trusted**. If it's compromised, it can log and reveal mappings between senders and receivers.

🔁 Two-Way Anonymous Communication

Chaum's original Mix only handles **one-way** messages. To allow **replies**, Alice includes in her message:

- A **return address** of sorts, created via:
  - A temporary **public key** `K2`.
  - An inner message `{K1, Alice}` encrypted with a **different Mix's** public key.

Bob, upon receiving Alice's message, uses this return address to craft a reply:

1. Encrypts his reply `M'` with `K2` (public key from Alice).
2. Sends `{K1, {M'}K2}` to the second Mix.
3. That Mix decrypts the outer layer, sees `K1`, uses it to encrypt the reply, and sends it to Alice.

Alice can now decrypt using her own keys. Bob **never learns who she is**.

🚧 Limitations

- **High latency**: Public-key operations are slow, batching causes delays.
- **Single point of failure**: If the Mix is down, all anonymity is lost.
- **Trust assumption**: If the Mix is compromised, the system fails.

🔁 Solution: Mix Cascades

To reduce reliance on one Mix, Chaum proposed **Mix Cascades**:

- Messages pass through **multiple mixes**, ideally run by **mutually distrusting entities**.

- As long as **one Mix is honest**, anonymity is preserved.

✅ Summary

Feature	Description
Purpose	Anonymize sender-recipient relationships over public networks
Mechanism	Public-key encryption + batching + shuffling
Strength	Prevents traffic correlation, strong against passive attackers
Limitation	High latency, assumes a trusted Mix
Extension	Mix cascades, anonymous reply mechanisms

Would you like a visual diagram or step-by-step encryption layering breakdown?

Can you explain the two-way anonymous communication in a bit more detail, like, I'm confused what is the K1 exactly here, and how does Bob know the return address of Alice?

Absolutely — this is one of the trickier but most elegant parts of Chaum's system: how to allow **Bob to reply to Alice without knowing who Alice is**, and yet still **get the reply to her securely and anonymously**.

Let me walk you through it carefully, step by step, with intuitive explanations.

🧭 Problem Setup: One-Way vs Two-Way

In **one-way anonymous mail**:

- Alice sends a message to Bob anonymously through a **Mix**.
- Bob receives the message but doesn't know who sent it.

That's fine for whistleblowing or reporting.
But what if Bob wants to reply to Alice?

That's where we need the **anonymous return address mechanism**, using `K1` and `K2`.

🔐 Step-by-Step: How Two-Way Anonymous Communication Works

🎁 Step 1: Alice Prepares the Message to Bob

She wants to:

1. **Send an anonymous message `M` to Bob.**

2. **Include a way for Bob to reply anonymously back to her**.

So she constructs the message like this:

```scss
{r1, {r0, M}pk(B), B}pk(Mix)
```

But now, inside `M`, she includes **two important components**:

💡 Inside M (Payload):

1. `{K1, A}pk(Mix')`
    - `K1` is a **random symmetric key**, generated by Alice.
    - `A` is Alice's identity (optional; could be a pseudonym).
    - This whole thing is encrypted under the **public key of another Mix**, say `Mix'`.
      👉 This is how Bob knows where to send the reply.
2. `K2`
    - A **temporary public key** generated by Alice, just for this conversation.
    - Bob will encrypt his reply under this key, and **only Alice has the private key**.

So:

> ✅ `K1` = symmetric key used by the second Mix to forward the reply to Alice.
> ✅ `K2` = public key used by Bob to encrypt the actual reply.

This whole package gets sent through the Mix to Bob.

🧾 Step 2: Bob Prepares a Reply

Now Bob has:

- The message `M` (from Alice)
- Inside it:
    - `K2` (public key)
    - `{K1, A}pk(Mix')` (an instruction for the return Mix)

So Bob prepares a **reply message** like this:

```less
{ {K1, A}pk(Mix'), {r2, M'}K2 }  --> sent to Mix'
```

- The **first part** is the same pink header Alice provided.
- The **second part** is the encrypted reply `M'`, encrypted with `K2` (Alice can decrypt it).

🧠 Step 3: Mix’ Forwards the Reply to Alice

Mix’ receives this reply message and knows it’s a **return message**, so it:

1. **Decrypts `{K1, A}`** using its private key.
2. It sees:
    - A destination `A` (presumably a public endpoint).
    - A symmetric key `K1`.
3. It then **encrypts the reply `{r2, M’}K2` using `K1`**:

```
{ {r2, M’}K2 }K1
```

4. Sends it to `A`.

🔓 Step 4: Alice Receives and Decrypts

When Alice receives the message:

- She uses her symmetric key `K1` to **decrypt the outer layer**.
- Then uses her private key for `K2` to **decrypt the reply from Bob**.

✅ Done! Alice got the reply, but **Bob still doesn’t know who she is**, and **Mix' only knows it sent a message to someone who holds `K1`**.

🔐 Why So Much Indirection?

Because:

- **Bob should not know Alice’s IP, email, etc.**
- **Mix' only sees a message going to “A” (an address that Alice controls)**.
- **Only Alice has both `K1` and the private key for `K2`**, so no one else can read the message.

📦 Real-World Analogy (Whistleblower)

- **Alice** is a whistleblower.
- **Bob** is a journalist.
- Alice sends her leak to Bob through a secure mailbox (Mix).
- Inside the leak, Alice includes a **return dropbox (Mix') address and decryption keys**, so Bob can reply—but he’ll never know who picked up the reply from the dropbox.

✅ Summary

Term	Role
`K1`	Symmetric key used by Mix' to forward the reply to Alice
`K2`	Alice's temporary public key for Bob to encrypt the reply
`Mix`	Trusted intermediary used to send the original message
`Mix'`	Return mix used to deliver the reply anonymously

Let me know if you want a diagram or walk-through with sample keys/messages!

ok thanks, now explain onion routing

You're welcome! Now let's dive into **Onion Routing**, a powerful technique for **low-latency anonymous communication**, most famously used in **Tor (The Onion Router)**.

I'll explain it step by step, based on **slides 14–15** and the **professor's detailed lecture**.

🧅 What Is Onion Routing?

Onion routing is a method where a message is wrapped in **multiple layers of encryption**, each layer corresponding to a hop (i.e., a router or relay) in the communication path.

At each hop:

- One layer of encryption is **peeled off** (like an onion skin).
- The message is forwarded to the **next node**.

> 🔐 This ensures that **no single router knows both the source and the destination**.

🎯 Key Goal

> To make it **impossible** for any intermediate node (or even an attacker who compromises one or more of them) to determine **who is talking to whom**.

🏗️ How It Works — Step-by-Step

Let's say **Alice wants to send a message to Bob** through a chain of 4 routers: **`R1` → `R2` → `R3` → `R4`**, where **`R4`** is the **exit node** that delivers the message to Bob.

🔑 Step 1: Key Establishment

Alice needs to:

- **Pick a path of routers** (say, R1–R4).
- **Establish a symmetric key** with each router:
  - `K1` with `R1`

- **K2** with **R2**
- **K3** with **R3**
- **K4** with **R4**

These keys are exchanged using **public-key cryptography**, but only during setup.

📦 Step 2: Onion Construction (Layered Encryption)

Suppose Alice wants to send a message **M** to Bob.

She wraps it like this:

```csharp
Layer 4: Encrypt M with K4  → {M}K4
Layer 3: Encrypt with K3    → {{M}K4}K3
Layer 2: Encrypt with K2    → {{{M}K4}K3}K2
Layer 1: Encrypt with K1    → {{{{M}K4}K3}K2}K1
```

So the final packet sent to **R1** looks like a big onion with 4 layers.

🔁 Step 3: Each Router Peels a Layer

Router	What It Sees	What It Does
R1	Encrypted blob + next hop (R2)	Decrypts outer layer using **K1**, sends to R2
R2	Smaller encrypted blob + next hop	Decrypts with **K2**, sends to R3
R3	Smaller encrypted blob + next hop	Decrypts with **K3**, sends to R4
R4	Final layer → message to Bob	Decrypts with **K4**, gets **{M}**, sends to Bob

At every step, the node:

- Only knows its **previous and next hops**.
- Cannot tell the **full path** or the **original sender**.
- Cannot read the message content unless it is the exit node (R4).

🔐 Security Properties

Property	How It's Achieved
Anonymity	No single node knows both sender and recipient
Unlinkability	Adversary can't link incoming and outgoing packets
Forward Secrecy	New keys per session, no long-term compromise

---

🧠 Professor's Analogy

The professor compared it to a **physical envelope** with **nested layers**:

- Each envelope has the address of the next delivery person.
- Only the final envelope has the actual message for Bob.
- Even if one person along the path is evil, they **only see one layer**.

He also said:

> "You better encrypt it first under R4's public key, then R3's, then R2's, then R1's — so only R1 can peel the outermost layer, and so on."

Hence the name: **onion routing**.

🕵️ What Can the Adversary See?

Even if an adversary controls one router:

- They can only see **who passed them the message**, and **to whom they are forwarding it**.
- They **cannot know** if they are the **first, middle, or last hop**—unless they are the entry or exit node.

🛠️ Practical Version: Tor

Tor (The Onion Router) implements onion routing with:

- A **telescopic route setup**: each hop is added **one at a time** over an encrypted channel.
- Uses **symmetric encryption** for data transfer (faster).
- **Three-hop circuits** by default:
  - Entry node
  - Middle relay
  - Exit node

🧅 Summary

Feature	Onion Routing
Goal	Anonymous low-latency communication
Mechanism	Layered encryption over multiple routers
Protection	Each router sees only one hop in the chain
Used in	Tor, anonymous messaging/browsing
Threat Resilience	Resists passive and some active attackers

Let me know if you want a diagram of the onion encryption/decryption layers or how Tor builds the circuit (telescopic setup)!

how does a router know where to forward the encrypted packet in onion routing?

Great question! This is **one of the most important technical points** in onion routing — and it's a bit subtle. Let's break it down clearly.

🌐 Problem:

In onion routing, a router receives a **partially encrypted blob**.
But **how does it know where to send it next**, if it **can't see the full routing path**?

✅ Answer:

Each router **learns only the identity of the *next hop*** from the **part of the message it decrypts**.

Let's walk through the steps using an example.

🧅 Step-by-Step — How Forwarding Info is Encoded

Suppose:

Alice wants to send a message through routers:
`R1 → R2 → R3 → R4` (where R4 is the **exit** node to Bob).

She prepares a multi-layer encrypted message like this:

For `R1`, she constructs:

```mathematica
{NextHop=R2, Key=K1,
  {NextHop=R3, Key=K2,
    {NextHop=R4, Key=K3,
      {FinalMsg to Bob}K4
    }pk(R3)
  }pk(R2)
}pk(R1)
```

Each layer contains:

- The address of the **next router**
- A fresh **session key** to communicate with it
- An **encrypted inner message** for that next hop

🔄 What Each Router Does

1. `R1` **receives the outermost encrypted blob:**
    - Decrypts it using its **private key**.
    - Learns:
        - **Next hop is R2**
        - Session key `K1` to use for this session
        - A smaller encrypted blob `{...}pk(R2)`
    - Uses this info to **forward the blob to** `R2`.
2. `R2` repeats the same:
    - Decrypts with its private key.
    - Learns:
        - Next hop is `R3`
        - Session key `K2`
        - Encrypted message for `R3`
    - Forwards it to `R3`.
3. And so on, until the exit node (`R4`) decrypts and delivers the actual message to Bob.

🔐 Key Insight

> Each router only knows:
>
> - Who sent the message to them (the **previous hop**)
> - Who to send it to (the **next hop**, revealed after decryption)
> - But **not the full path** or the original sender/recipient

This design gives:

- **Anonymity**: no node sees the whole chain.
- **Forward secrecy**: each hop uses a different key.
- **Minimal trust**: as long as **one node is honest**, privacy is preserved.

🧠 Professor's Analogy

The professor described this as each router **"peeling off a layer"** of the onion:

- Inside that layer is the address of the next router.
- Once decrypted, the router knows **just enough** to pass the message forward — no more.

✅ Summary

Router Learns	How?
Next hop	From decrypted layer of the message
Session key	Included in encrypted layer
Encrypted payload	Passed along without being read
Other hops (past/future)	**Not known**

Would you like a simple ASCII-style diagram of this process?

explain tor everything upto slide 25

Great — let's now walk through **Tor (The Onion Router)**, using both **slides 16–25** and the **professor's detailed lecture**.

We'll break it into digestible parts:

🧅 Slide 16 – What is Tor?

Tor is a **second-generation onion routing network** designed for:

- **Low-latency anonymous communication**
- Use cases like **web browsing**, **chat**, **messaging**, etc.

  🔍 **Tor = The Onion Router**

Key Features:

- Runs since **2003**
- Used by **millions** of users worldwide
- Thousands of volunteer nodes (routers)
- Freely available — e.g., **Tor Browser**

🔗 Slide 17–19 – Tor Circuit Setup (1–3)

Tor builds an anonymous **"circuit"** through three relays:

1. **Entry node** (a.k.a. "Guard")
2. **Middle node**
3. **Exit node**

Let's understand the process in 3 steps:

🔐 (1) Setup with First Router (Entry Node)

- The client (e.g., Tor Browser) contacts the **first node (R1)** using **public-key cryptography**.
- They negotiate a **shared symmetric key**.
- This forms a **secure channel** (like a VPN tunnel).

🔐 (2) Extend to Second Router

- The client now uses the first hop to **extend the circuit to a second node (R2)**.
- Still using public-key crypto, it:
  - Creates a shared key **K2** with **R2**
  - Sends the setup message **through R1**

Now there's an encrypted tunnel from the client → R1 → R2.

🔐 (3) Extend to Third Router (Exit Node)

- Same idea:
  - Through the existing tunnel, the client negotiates a shared key **K3** with **R3** (exit node).
- The client now has:
  - Key **K1** with R1
  - Key **K2** with R2
  - Key **K3** with R3

Result:

> A **3-hop encrypted circuit**: client → R1 → R2 → R3
> All hops use **symmetric encryption**, which is **fast**.

This circuit is reused for multiple TCP streams (e.g., web tabs).

🌍 Slide 20 – Using the Circuit

Once the circuit is ready:

- The client encrypts a message for the destination (e.g., visiting a website).
- The message is **wrapped in layers**:
  - Innermost layer: for R3 (exit)
  - Middle layer: for R2
  - Outermost: for R1

At each hop:

- A router decrypts one layer

- Learns **only the next hop**
- Forwards the remaining blob

🕵️ Slides 21–24 – What Does the Adversary Know?

The adversary may be watching different parts of the system. Let's consider each case:

👀 Adversary Observing Alice (Client)

- Sees that Alice is connecting to **some Tor node**
- But **doesn't know the final destination**
- Cannot see the actual data due to encryption

> 🔐 Tor hides *what* Alice is doing, even if it doesn't hide *that* she's using Tor.

👀 Adversary Inside the Tor Network (e.g., Watching R2)

- Can only see:
  - The **previous hop** (R1)
  - The **next hop** (R3)
- Doesn't know if it's the **first**, **middle**, or **last** node in the circuit

> 🤷 Can't trace the message backward or forward.

👀 Adversary at the Exit Node (R3)

- This is the **most dangerous position**.
- Can see the **final destination** and possibly the **plaintext** if the traffic is not encrypted (e.g., HTTP).
- But:
  - Doesn't know **who the original sender was**
  - Only knows it came from R2

🛡️ To protect against this:

> ✅ Always use **HTTPS/TLS** when browsing via Tor.

🧠 Slide 25 – Tor Management Issues

✅ Multiple Applications per Circuit:

- Tor allows **many TCP streams** to share the same circuit (e.g., multiple tabs in Tor Browser).
- This improves performance and obfuscation.

✅ Anyone Can Run a Tor Node:

- No root/admin privilege required.
- Encourages more people to participate.
- **More nodes = better anonymity for everyone**.

🗂️ Directory Servers:

- Special servers keep track of:
  - Available Tor routers
  - Their **IP addresses**, **public keys**, **reliability**
- Clients download this directory to pick relays.

⚠️ Threat: Sybil Attack

- An attacker creates **many fake Tor nodes** to try to **control a large portion of the network**.
- If successful, could deanonymize users.
- That's why:
  - Directory servers are **trusted**
  - Their **public keys ship with the Tor software** (hardcoded)
  - New nodes must **build reputation** over time before being trusted

---

✅ Summary up to Slide 25

Concept	Description
Tor Network	Uses onion routing for anonymous communication
Circuit Setup	Telescopic, 3-hop path with symmetric keys (K1, K2, K3)
Encryption	Each message is layered and peeled at each hop
Adversary Views	Sees only partial info; full traceability is prevented
Directory Servers	Maintain trusted lists of Tor nodes; vulnerable to Sybil if compromised
Threats	Exit node snooping, Sybil attack, DoS on nodes
Protection Advice	Use HTTPS, avoid entering personal info, diversify circuits for isolation

Let me know if you'd like to go into **telescopic setup** or the **location-hidden services** next!

whats this directory server

Great question! The **Directory Server** is a crucial part of how the Tor network functions securely and reliably — it's what **enables Tor clients to discover trusted, up-to-date Tor nodes**.

Let me break it down clearly.

📖 What is a Directory Server in Tor?

A **Directory Server** is a **trusted node** in the Tor network whose job is to:

- Maintain a **list of active Tor routers (relays)**.
- Provide their:
  - **IP addresses**
  - **Public keys**
  - **Geographic locations**
  - **Performance metrics** (uptime, bandwidth, etc.)
- Share this information with **Tor clients** (e.g., your Tor Browser) so they can **build secure circuits**.

  🧭 Think of it like a **"yellow pages"** for Tor nodes.

🏗️ Why Are Directory Servers Needed?

1. 🧱 Without a directory, **clients wouldn't know which routers exist**.
2. 🧾 Each Tor client **downloads a "consensus" document** from directory servers at startup.
3. 📡 This document helps the client pick:
   - A reliable **entry node**
   - A random **middle node**
   - A good **exit node**

🔐 Why Do We Trust Directory Servers?

- They are **run by trusted organizations**, like the **Tor Project** and partners (e.g., academic groups, privacy NGOs).
- Their **public keys are hardcoded into the Tor software**.
- Clients only trust a node list that has been **digitally signed by the majority of these directory authorities**.

  📜 This digital signature ensures **integrity** and **authenticity** of the directory information.

⚠️ What If a Directory Server Is Compromised?

This is a big **Point of Vulnerability (PoV)**:

- If attackers **poison the directory**, they could:
  - Insert **malicious nodes** (e.g., all three in your circuit)
  - Perform a **Sybil attack** (control a large part of the network)

- Attempt to **deanonymize users**

That's why:

- There are only a **small number of directory servers (around 10)**.
- They use a **quorum voting system** — majority must agree on the directory contents.
- They're under **tight operational controls**.

🧠 Professor's Point

In the lecture, the professor emphasized:

> "Directory servers' keys ship with the Tor code — that's a **major point of vulnerability**."

That means:

- If a state-level attacker compromises **enough directory servers**, they could distribute **malicious Tor nodes**.
- This could break anonymity for many users.

✅ Summary

Feature	Description
Role	Maintains up-to-date list of Tor relays
Used for	Circuit construction by clients
Contents	IP, public key, location, uptime, bandwidth
Trust model	Few trusted servers, hardcoded public keys
Risk	Compromise = poisoned network (Sybil attack risk)
Defenses	Digital signatures, quorum voting, reputation checks

Let me know if you'd like to explore **how Tor clients validate and download the consensus**, or how Sybil attacks can be countered!

explain Location Hidden Server upto slide 31

Absolutely — the concept of a **Location Hidden Server** is a fascinating and powerful use case of Tor. Let's walk through everything up to **slide 31**, using both the **slides (26–31)** and the **professor's detailed lecture**.

🎯 Goal: What Is a Location Hidden Server?

A **Location Hidden Server** is a service (like a website or marketplace) that runs **anonymously** on the Tor network. Its physical **location and operator's identity are unknown** to both users and attackers.

💡 Why Use It?

A hidden service allows you to:

- Deploy a **server** (e.g., a website) that:
  - Can be **accessed anonymously**
  - Is **resistant to censorship**
  - Can **survive DoS attacks**
  - Cannot be physically found or shut down easily

📌 Example: The infamous **Silk Road** marketplace was a Tor hidden service.

🧅 Key Concepts

- **Both client and server are anonymous**
- **No IP address of the server is revealed**
- Communication occurs entirely **within the Tor network**
- URLs are `.onion` addresses — generated cryptographically

🏗️ How It Works — Step-by-Step

🧩 Step 1 – Server Sets Up Introduction Points

(slide 28)

- The hidden service (e.g., Bob's secret site) creates **Tor circuits** to a few Tor nodes called **Introduction Points**.
- These are just **regular Tor relays**, but now they're acting as **stable entry points** into the service.
- The service **publishes a descriptor**, which includes:
  - Its public key
  - List of introduction points
- This descriptor is stored in a **Distributed Hash Table (DHT)** in the Tor network (via directory services).

  So now anyone can access this server via its `.onion` address, which points to those intro points.

🧍 Step 2 – Client Fetches Descriptor

- The client (e.g., Alice) wants to connect to Bob's hidden service.
- She gets Bob’s `.onion` address and:
  - Looks it up in the Tor directory
  - Downloads the service descriptor
  - Learns about Bob’s intro points

🔗 Step 3 – Client Builds Rendezvous Circuit

(slide 29)

- Alice selects a **random Tor node** to act as a **Rendezvous Point**.
- She creates a Tor circuit to this node (just like any other circuit).
- Then she sends a message to **one of Bob’s intro points** that includes:
  - The address of the **rendezvous point**
  - A temporary one-time key for secure communication
  - Any **authorization data** (if the service is private)

🤝 Step 4 – Bob Connects to Rendezvous Point

- The intro point **notifies Bob**: “Hey, someone wants to talk to you.”
- If Bob agrees, he:
  - Creates a **Tor circuit** to the rendezvous point
  - When his message arrives, the rendezvous node connects the two circuits

🎉 Now:

- Neither Alice nor Bob knows each other's location.
- They communicate through **two separate Tor circuits**, joined at the **rendezvous point**.

🛡️ What Does This Protect?

Threat	Hidden Service Protection?
Server IP address exposed?	❌ No – hidden in Tor
DNS lookup reveals server location?	❌ No – no DNS, just `.onion`
DoS attack on server?	✅ Harder due to indirection
Censorship/Blocking?	✅ Can’t block what you can’t locate

🎭 Slide 30 – Real Example: Silk Road

- **Silk Road** was a Tor hidden service that operated a **black market**.
- It sold:
  - Drugs
  - Stolen electronics
  - Fake IDs
  - Even murder-for-hire services (allegedly)

👩‍⚖️ Slide 31 – The Shutdown of Silk Road

Ross Ulbricht ("Dread Pirate Roberts") was arrested in 2013.

Interestingly:

> The shutdown was **not due to Tor breaking**, but due to **operational mistakes**.

Professor's Highlights:

- Ross posted a Stack Overflow question under his **real name**, asking about `.onion` usage in PHP.
- Changed his username seconds later to "frosty", but logs had already recorded the post.
- The **Silk Road server's encryption key** also contained "frosty@frosty", linking him to the service.
- Also, a **package of fake IDs** was mailed to him — possibly a sting operation.

🧠 Summary of Slides 26–31

Step	Action	Who Does It
1	Create intro points + descriptor	Server (Bob)
2	Lookup `.onion` address	Client (Alice)
3	Create rendezvous point + notify	Alice → intro point
4	Join at rendezvous point	Bob ⟷ Alice via Tor

Benefit	Why It Matters
No IP address exposed	True anonymity for server operator
Resistant to blocking	Can't filter what you can't locate
Privacy for both parties	Neither side knows the other's identity
Censorship resilience	Great for journalists, activists

Let me know if you want a visual diagram or want to dive into **how the rendezvous key exchange** works!

explain the Dining Cryptographers and onwards topics

Great! Let's now explore the final and most intellectually rich section of the anonymity lecture: **The Dining Cryptographers Problem** and its generalizations, which begin on **slide 33** and continue through to **slide 38**.

This portion focuses on achieving **information-theoretic anonymity** — meaning, anonymity that holds even against an attacker with infinite computing power.

🍽️ Slide 33 – What is the Dining Cryptographers Problem?

This problem was introduced by **David Chaum in 1988**, in the paper:

> "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability"

🎯 Goal:

To let someone **send a message anonymously in a group**, such that:

- The **message becomes public**, but
- **No one can determine who sent it**, **not even with infinite computation**.

This provides **stronger anonymity** than Tor or mixnets, which rely on cryptographic assumptions.

🧠 The Story Setup

Three cryptographers are dining. The check is paid by:

1. The **NSA**, or
2. **One of the cryptographers**, who wants to remain anonymous.

The goal is:

> Determine **whether the NSA paid or one of us did**, but **don't reveal who**.

🧪 Slide 34 – The Protocol

Each person:

1. **Flips a coin** and **shows it to their neighbor on the left**.
2. Each person now sees **two coins**:
   - Their own
   - The one from their right neighbor
3. Each person **announces** whether the two coins they saw were the **same** or **different**:
   - Unless **they are the payer** — then they **lie** (say the opposite).

🧮 Interpretation of Results:

- Count the number of people who said **“same”**.
- If the count is:
  - **Even (0 or 2)** → One of the cryptographers paid.
  - **Odd (1 or 3)** → The NSA paid.

🔐 Slide 36–37 – Why This Works

Imagine you are a **non-payer**:

- You know your coin flip.
- You see your neighbor's coin.
- You know what you said — and what the others said.
  But still:

  You **can’t tell who lied**, because **your view is consistent with multiple possibilities**.

This makes it **impossible to link the announcement of “same” or “different” to a specific payer**.

📈 Slide 38 – Generalization: Superposed Sending

To scale the idea to **N participants** and **allow sending real messages**, we use **superposed sending** (also called a DC-net: Dining Cryptographers Network).

🧩 How it works:

1. Each participant generates a **random bit**, and **shares it with one neighbor**.
2. Each participant knows:
   - Their own bit
   - Their neighbor’s bit
3. Each participant:
   - Announces the XOR of their two bits, unless they want to send a message bit `m`
   - If sending, they announce:
     `their_bit XOR neighbor_bit XOR m`
4. XOR all announcements together:
   - Random bits cancel out (appear twice)
   - The message bit `m` appears once

✅ Final XOR = message bit

🛡️ Key Properties

Property	Achieved By
Untraceability	Random sharing and XOR canceling
Sender anonymity	Because anyone could have added the message bit
Information-theoretic security	No assumptions about computation

---

❗ Limitations

1. **Collisions**: If two people try to send at once, the XOR gets corrupted.
2. **Scalability**: Every user must coordinate random bits with others.
3. **No sender authentication**: Anyone can inject bits.

Still, it's a **theoretical gold standard** for perfect anonymity.

🧠 Professor's Intuition

- The protocol **does not rely on public-key cryptography** or hard math problems.
- It gives **unconditional anonymity**.
- Hard to scale in practice, but **beautiful in design**.
- It's like a **group confession booth** — someone admits something, but no one knows who.

✅ Summary: Dining Cryptographers & Beyond

Topic	Key Idea
Dining Cryptographers	3 participants reveal if one paid, but not who
Superposed Sending	Allows actual message transmission with anonymity
XOR mechanism	Cancels noise, reveals only one message bit
Security level	Information-theoretic (even NSA with infinite compute can't break it)
Limitations	Collisions, coordination, impractical for large N

Let me know if you want a step-by-step walkthrough with actual bits, or a small code simulation!

So I have few confusion regarding the superfast sending method. Firstly, each user will XOR his own bit with neighbor's bit, but someone can XOR with left neighbor's bit, someone can XOR with his right neighbor's bit. So eventually the claim that all the bits will cancel out because of appearing twice and only the remaining message bit M will appear only once. I don't think how does that happen. Also, what if multiple users want to send messages? Then I think XOR of multiple messages will remain. So how will that be handled?

Fantastic observations! These are exactly the kinds of nuanced questions that arise when studying **DC-nets (Dining Cryptographers networks)** and **superposed sending** in detail. Let's walk through your concerns carefully.

❓Confusion 1: "Everyone XORs their bit with their neighbor's – but which neighbor? Left or right? What ensures bits cancel out?"

✅ Clarification:

Each **pair of users** shares a **random bit**, and both participants know it.

Let's denote:

- Users: **`A`**, **`B`**, **`C`**, ... up to **`N`**
- Shared random bits:
  - **`rAB`** is shared between A and B
  - **`rBC`** is shared between B and C
  - ...
- Important: **rAB = rBA**, both A and B know it

So in total, for **`N`** users, there are **`N choose 2`** shared random bits if fully connected, or just **`N`** if ring-based (e.g., only with left/right neighbors).

💡 What Each User Broadcasts

User **`U`** computes:

```ini
broadcast = XOR of all shared bits they hold
```

Let's say in a **ring topology**, where each user shares a bit with left and right neighbors:

User A:

- Shares **`rAB`** with B
- Shares **`rAN`** with N (left neighbor)

Then A computes:

```ini
mA = rAB ⊕ rAN
```

All users do the same.

Now, if we XOR **all the broadcasts** together:

```scss
(mA ⊕ mB ⊕ mC ⊕ ... ⊕ mN)
```

Each shared bit (e.g., **`rAB`**) **appears twice**, once in A's broadcast, once in B's.
Therefore:

> Every random bit appears exactly twice → XOR cancels them all.

✅ So only what remains is: the message bit (if any)

Now, if **one person** (say, Alice) wants to send a 1-bit message **`m`**, she **flips** one of the shared bits *before broadcasting*. This changes the final sum:

```scss
(rAB ⊕ rAN) ⊕ m   ← Alice's broadcast
```

Total XOR result becomes:

```python
(⊕ all shared bits) ⊕ m = 0 ⊕ m = m
```

This is why **the message bit appears once**, and everything else cancels.

❓ Confusion 2: "What if multiple people send messages at once?"

You are **exactly right** — this is a limitation.

🎯 XOR is linear:

So if:

- Alice sends message **`m1`**
- Bob sends message **`m2`**
- Then the final result is:

  ```nginx
  m1 ⊕ m2
  ```

🚨 Problem:

- The group knows **some message was sent**, but not **how many** or **what each one was**.

✅ How Is This Handled?

1. Collision Avoidance (Basic Protocol Assumption):

- **Only one user is allowed to send per round**
- Senders take turns or coordinate via a higher-layer protocol

⚠️ Downside: requires scheduling or coordination

---

2. Collision Detection:

- If multiple users send, the XOR becomes **garbled**.
- You can **detect** that a collision happened (e.g., non-zero result when none was expected).
- Then the group retries with some **randomized backoff protocol**, like in Ethernet or ALOHA.

---

3. Advanced DC-net Variants:

There are more complex versions that:

- **Split messages across multiple rounds**
- Add **cryptographic commitments**
- Use **zero-knowledge proofs** to avoid disruption attacks

But these are more computationally expensive and beyond the basic protocol.

✅ Summary of Your Questions

Concern	Answer
How do bits cancel properly?	Each shared bit is XOR'd by **both parties**, appears twice → cancels
Left or right neighbor?	Doesn't matter — **both sides XOR the shared bit**, so it's symmetric
What if multiple users send?	XOR of messages → m1 ⊕ m2 → garbled; collisions must be **avoided or detected**
How to resolve collisions?	Retry with **scheduling**, **randomized backoff**, or use **more advanced DC-nets**

Let me know if you want to walk through a concrete 3-user example with bit values — it can help solidify the XOR cancellation intuitively.

Recall the Dining Cryptographers problem setting with 3 diners. From the point of the view of the non-payer,
answer the following:
A) What happens if all three declare "SAME" at the end of the protocol.
B) What happens if one says "SAME" and two say "DIFFERENT"?

Great! Let's revisit the **Dining Cryptographers Problem** with **3 diners** and answer the two cases based on the protocol logic and the **view of the non-payer**.

🔁 Quick Recap of the Protocol:

- Each diner flips a coin and shows it to their **left neighbor**.
- Each diner sees **two coins**: their own + their right neighbor's.
- Each diner **announces "SAME" or "DIFFERENT"** based on whether the two coins they saw match.
- **But**: if the diner **paid for the meal**, they **lie** — they say the opposite.

📌 Rule for Determining Who Paid:

- **If the number of "SAME" declarations is even** → One of the diners paid
- **If the number of "SAME" declarations is odd** → NSA paid

A) ❓All Three Say "SAME"

- Count of "SAME" = 3 → **Odd**
- According to the rule: **NSA paid**.

✅ So from the non-payer's view:

- The NSA paid.
- No one lied.
- All diners told the truth.
- Since the non-payer knows they didn't pay, and everyone said "SAME", no contradiction.

B) ❓One Says "SAME", Two Say "DIFFERENT"

- Count of "SAME" = 1 → **Odd**
- According to the rule: **NSA paid**

But here's the catch:

- NSA is supposed to pay **only if no one lies**.
- Yet, in this case, the pattern of outputs suggests someone might have lied (since the outputs don't match up cleanly with coin observations).

👀 From the non-payer's point of view:

- They know:
  - Their own coin
  - The coin shown to them

  - Their own declaration (“SAME” or “DIFFERENT”)
- They see the other two declarations (1 “SAME”, 2 “DIFFERENT” total).

🧠 What Can They Infer?

- They **know they didn’t pay**.
- But since the **count is odd**, the protocol says: **NSA paid**, so:
  - **No one is supposed to lie**.
  - The two people who said “DIFFERENT” must have **seen different coins**, honestly.
  - The one who said “SAME” must have seen matching coins.

✅ So this is a **valid situation** under the protocol.

> The non-payer concludes that **NSA paid**, and no one lied.

✅ Final Answers

Case	Total SAMEs	Who Paid?	What Can Non-Payer Infer?
A) All said “SAME”	3 (odd)	NSA	NSA paid; no one lied; output is consistent
B) One “SAME”, two “DIFFERENT”	1 (odd)	NSA	NSA paid; no one lied; coin views led to differing reports

Let me know if you'd like a concrete coin example for either case!

How does a merchant (Bob) use the Tor anonymity network to set up a location-hidden service? How does a
customer (Alice) learn about Bob’s location-hidden service and how does she use it? Carefully list all necessary steps.

Excellent question. Let’s walk through the **complete lifecycle** of how a merchant like **Bob** sets up a **Tor location-hidden service**, and how a customer like **Alice** finds and connects to it — using both the **slides (28–29)** and the **professor's lecture**.

We'll explain this in two roles:

- **Bob (Merchant)**: wants to **anonymously host a service**
- **Alice (Customer)**: wants to **access that service anonymously**

🧱 PART 1: How Bob Sets Up a Location-Hidden Service

🔧 Step 1: Bob chooses some Introduction Points

- Bob picks **several Tor nodes** to act as his **Introduction Points**.

- These are regular Tor relays with which he builds **semi-permanent Tor circuits**.

🧾 Step 2: Bob creates a Service Descriptor

- This descriptor includes:
  - The **public key** of the service
  - The **.onion address** (derived from the public key)
  - The list of **introduction point addresses**
- The `.onion` address is self-authenticating: it's a hash of the public key.

🌍 Step 3: Bob uploads the descriptor

- The service descriptor is published to the **Tor distributed hash table (DHT)**.
- This acts like a **lookup directory** within Tor.

✅ At this point, Bob is ready. His service is **accessible via its .onion address**, but his **IP address and location are hidden**.

🧭 PART 2: How Alice (Customer) Connects to Bob's Hidden Service

🔍 Step 4: Alice learns Bob's .onion address

- She may get this address from:
  - A friend
  - A public website
  - Some other communication (e.g., a hidden wiki)

📥 Step 5: Alice fetches Bob's service descriptor

- Alice queries the **Tor DHT** using Bob's `.onion` address.
- She retrieves:
  - The list of **Introduction Points**
  - Bob's **public key**

🔁 Step 6: Alice creates a Rendezvous Point

- Alice randomly selects a Tor node to act as a **rendezvous point**.
- She builds a Tor circuit to it and **generates a one-time secret** for secure communication.

📦 Step 7: Alice contacts one of Bob's Introduction Points

- Through one of Bob's intro points, Alice sends:
  - The **address of the rendezvous point**

- Her one-time **rendezvous key** (for encrypted reply)
- Optional **authorization**, if Bob's service is private

🔄 Step 8: Bob is notified by the Introduction Point

- The intro point forwards Alice's request to Bob over their semi-permanent circuit.

🤝 Step 9: Bob connects to the Rendezvous Point

- Bob uses the information Alice sent to:
  - Establish a circuit to the **rendezvous point**
  - **Complete a handshake** using the rendezvous key

🔐 Step 10: Alice and Bob communicate anonymously

- They now have two Tor circuits:
  - Alice → Rendezvous Point
  - Bob → Rendezvous Point
- The **rendezvous point bridges them together**
- **All traffic is encrypted**, and neither party knows the other's IP address

---

✅ Summary of Steps

Actor	Step	Action
Bob	1	Chooses multiple introduction points
	2	Creates a service descriptor with pubkey + intro points
	3	Publishes it to the Tor DHT
Alice	4	Learns the `.onion` address
	5	Fetches Bob's descriptor from the DHT
	6	Selects a rendezvous point and builds a circuit to it
	7	Sends rendezvous info to one of Bob's intro points
Bob	8	Gets the request and connects to the rendezvous point
Both	9	Exchange messages anonymously via rendezvous point

This setup allows **anonymous publishing, resistant to censorship, surveillance, or physical shutdown**, and is a powerful tool for privacy.

Let me know if you want a diagram of the connections or want to walk through the encryption handshake.

Suppose Alice and Bob each have a TOR Hidden Service: THS-Alice and THS-Bob. Alice lives in a place where
alcohol is taboo and Bob lives in a place where humor is forbidden. Alice sells cocktail recipes, and Bob sells
funny (but sometimes offensive) cartoons. (Note that both types of "goods'' are digital.) Design a scheme so
that Alice and Bob could perform *barter* using their respective Tor Hidden Services, i.e., instead of paying
each other in some form of currency (money), Alice would "exchange" one cocktail recipe for one of Bob's
cartoons. Try to make your protocol as fair as possible. Can it ever be perfectly fair? "Perfectly fair" means
that – if the protocol breaks down at some point, or one of them aborts – neither party has any advantage over
the other.

This is a fascinating problem that combines **Tor hidden services**, **barter**, and **fair exchange protocols** — all in a high-censorship setting. Let's walk through a secure and anonymous **barter protocol** between **THS-Alice** and **THS-Bob** for digital goods, aiming for **maximum fairness**.

🎯 Goal

Enable Alice and Bob to exchange:

- 🍹 A **cocktail recipe** from Alice
- 🤡 A **cartoon** from Bob
  via their respective **Tor Hidden Services (THS-Alice, THS-Bob)**
  without using currency, and ideally:

  🔐 **Fairly** — either both parties get what they want, or neither does.

⚠️ Fair Exchange Is Hard

> In **asynchronous systems**, **perfect fairness is impossible** without a trusted third party (due to the possibility of aborting mid-protocol).

We can only strive for:

- **Practical fairness** (cheating is detectable and discouraged)
- **Abortable protocols** with minimal leakage

Let's design such a scheme.

🛠️ Design: Fair Barter via Encrypted Commitments

We'll use a protocol based on:

- **Symmetric encryption**
- **Commitments**
- **Atomic release**

✅ Step-by-Step Protocol

🧑‍🍳 Step 1: Alice prepares her cocktail recipe

- Alice has a file `recipe.txt`
- She generates a random **symmetric key** `K_A`
- Encrypts her recipe:
  `C_A = Enc(K_A, recipe.txt)`

She sends `C_A` (encrypted recipe) to Bob via **THS-Alice**

🤡 Step 2: Bob prepares his cartoon

- Bob has a file `cartoon.jpg`
- Generates a random **symmetric key** `K_B`
- Encrypts his cartoon:
  `C_B = Enc(K_B, cartoon.jpg)`

He sends `C_B` (encrypted cartoon) to Alice via **THS-Bob**

🔁 Step 3: Key Exchange (Atomic Release)

At this point, each has:

- Encrypted file from the other
- Their own secret decryption key

Atomic Key Release Strategy:

1. Alice sends `K_A` to Bob
2. Bob verifies:
   - Decrypts `C_A`
   - Checks that it contains the cocktail recipe
   - If satisfied, sends `K_B` to Alice

Alice then decrypts `C_B` to get the cartoon.

🛡️ What Could Go Wrong?

❌ Problem: Bob gets `K_A`, but doesn't send `K_B`

- Bob gets the recipe for free
- Alice gets nothing

> 🔄 This is the **core fairness problem** in digital barter.

🔄 Optional Enhancements for Better Fairness

1. Use a cryptographic hash commitment

Before sharing any data:

- Alice sends `H_A = H(recipe.txt)`
- Bob sends `H_B = H(cartoon.jpg)`

This way, each party can verify content integrity after decryption.
It doesn't solve fairness, but deters **post-decryption cheating**.

---

2. Use a "Fair Exchange Escrow" server (trusted or federated)

- They send encrypted data to an **escrow relay service**
- Escrow service **atomically releases both keys**
- Neither side can abort after committing

This requires a **semi-trusted third party**, so may not be desirable in your setting.

---

3. Gradual Release Protocol

Split each file into `n` chunks:

- Each chunk is encrypted separately
- Chunks + keys exchanged in rounds:
  - A sends chunk + key 1
  - B sends chunk + key 1
  - Repeat...

> 🧠 Advantage: Early abort yields **partial leakage only**

✅ Final Protocol Summary (Simple Version)

Step	Alice (THS-Alice)	Bob (THS-Bob)
1	Encrypts recipe with key **K_A**	Encrypts cartoon with key **K_B**
2	Sends **C_A** to Bob	Sends **C_B** to Alice
3	Sends **K_A** to Bob	Verifies, then sends **K_B**
4	Decrypts **C_B**	Decrypts **C_A**

✅ Practical anonymity and confidentiality
⚠️ Not perfectly fair (last-mover advantage)

❗ Can It Be Perfectly Fair?

❌ **No**, not without:

- Trusted third party
- Hardware-enforced atomicity
- Blockchain-based smart contracts (not practical in .onion context)

You can only achieve:

- **Detectable cheating**
- **Discouragement of misbehavior**
- **Partial fairness via chunking or delay**

Let me know if you'd like a version with **time-lock encryption**, **blockchain enforcement**, or **reputation-based deterrents**!