CS203 Gene's Class - SSL TLS Overview

LEC5-SSLTLS - Compatibility Mode.pdf
PDF

LEC5-SSLTLS

read it very carefully

I have **carefully read** your uploaded document: **"LEC5-SSLTLS"**.

Here's a **summary of the key points** from it:

🔵 SSL/TLS

- **SSL (Secure Sockets Layer)**:
  - Versions: flawed v2, good v3.
- **TLS (Transport Layer Security)**:
  - TLS 1.0 = SSL 3.0 with minor changes (RFC 2246).
  - OpenSSL is a key implementation.
- **Provides**:
  - Security **at the transport layer** (above TCP).
  - Used mainly for secure communication over HTTP (HTTPS).
- **SSL Architecture**:
  - **Record Protocol**: Secure, reliable channel.
  - **Upper Layer Protocols**: Handshake, Alert, Change Cipher Spec.

🔵 SSL Record Protocol

- **Functions**:
  - **Confidentiality**: Encrypt data (e.g., AES-256).
  - **Integrity/Authentication**: HMAC (e.g., HMAC-SHA256).
  - Fragments, MACs, pads, encrypts, and adds headers.
- **Session**:
  - Defines parameters like keys and certificates.
- **Connection**:
  - Has states like nonces, keys, IVs, sequence numbers.

🔵 SSL Handshake Protocol

- **Goals**:
  - Entity authentication (server always, client optional).
  - Fresh shared secret establishment.
  - Secure ciphersuite negotiation.
- **Key Exchange**:
  - RSA (common), Diffie-Hellman (fixed/ephemeral/anonymous).
- **Authentication**:
  - RSA-based typically (ability to decrypt proves server's identity).
- **Key Derivation**:
  - pre_master_secret → master_secret → key_block → MAC/encryption keys.
- **Typical Handshake Run**:
  1. ClientHello
  2. ServerHello, ServerCertChain, ServerHelloDone
  3. ClientKeyExchange, ChangeCipherSpec, ClientFinished
  4. Server ChangeCipherSpec, ServerFinished
- **Session Resumption** and **Re-keying** are supported.

🔵 Other SSL Protocols

- **Alert Protocol**: Error reporting.
- **Change Cipher Spec Protocol**: Switch to new cipher suite.

🔵 SSL vs TLS

- **TLS1.0** = **SSL3.0 + minor tweaks**:
  - Use of HMAC.
  - Different key derivation using a PRF (Pseudo Random Function).
  - Better padding, more alert codes, etc.

🔵 Applications

- **E-commerce**: Securely transmitting credit card information.
- **Banking**: Sometimes uses client certificates.
- **Concerns**:
  - Poor server practices (e.g., insecure storage).
  - Client misunderstanding security warnings.

- Client software not verifying certificates properly.

🔵 SSL/TLS Known Vulnerabilities

- **Bad random number generators** → Weak keys (Goldberg and Wagner, 1996).
- **Padding oracle attacks** → Recover plaintext (Canvel et al., 2003).
- **Timing attacks** → Extract server private key (Boneh and Brumley, 2003).

🔵 SSH (Secure Shell)

- **SSH-1** (flawed), **SSH-2** (improved).
- **Application layer** security (unlike SSL/TLS at transport layer).
- **SSH-2 Architecture**:
  - Transport Layer Protocol (server authentication).
  - Authentication Protocol (client authentication).
  - Connection Protocol (multiplexed sessions).
- **Key Establishment**:
  - Diffie-Hellman key exchange.
  - Server authentication using RSA or DSS.
- **Common Algorithms**:
  - HMAC-SHA1/MD5, 3DES, AES.
- **SSH Port Forwarding**:
  - Redirects local TCP ports through SSH tunnel securely.
  - Example: Securing email communication.

🔵 Comparison: IPSec vs SSL/TLS vs SSH

- **All**:
  - Establish shared keys → Secure channel.
  - Protect ciphersuite negotiation.
- **Differences**:
  - **Network layer** (IPSec) vs **Transport** (SSL/TLS) vs **Application** (SSH).
- **Common Weaknesses**:
  - Bad implementations.
  - Poor platform security (rootkits, keyloggers).
  - Lack of user education (clicking through cert warnings, phishing).

🔵 Final Quote

> "Using encryption on the Internet is like arranging an armored car to deliver credit-card information from someone living in a cardboard box to someone living on a park bench."

If you want, I can also:

- Create **flashcards** for quick revision ✅
- Draw a **diagram** for SSL/TLS handshake or SSH architecture ✅
- Help you **compare SSL/TLS and SSH** in more detail ✅
- Help **explain flaws** like Padding Oracle or Timing Attacks intuitively ✅

Would you like me to prepare any of these for you? 📚🚀
(Just tell me!)

tell me which things are computed using master_secret?

Good question — let's be **very precise**:

The following things are computed **using the `master_secret`** in SSL/TLS:

---

1. Key Block (key material)

- From the `master_secret` and the client/server nonces, a **key_block** is derived.
- This **key_block** is then **split** into:
  - **MAC keys** (for integrity check at Record Layer)
  - **Encryption keys** (for confidentiality at Record Layer)
  - **Initialization vectors (IVs)** (if using block ciphers like AES-CBC)

🔵 **Purpose**: These keys are used by the **SSL Record Protocol** to secure the application data.

---

2. MAC in Finished Messages

- During the **Handshake**, after exchanging all key materials, the **ClientFinished** and **ServerFinished** messages are sent.
- Each **Finished message** contains a **MAC** computed over **all prior handshake messages**.
- This MAC is computed **using keys derived from the `master_secret`**.

🔵 **Purpose**: It ensures **no tampering** occurred during the handshake.
If the MAC is wrong, the handshake is aborted.

Quick Visual:

```vbnet
pre_master_secret → (hashing with nonces) → master_secret
master_secret → (more hashing with nonces) → key_block
key_block → [MAC key | Encryption key | IVs]

(master_secret also used to compute MACs in ClientFinished/ServerFinished)
```

So, in one line:

> The `master_secret` is used to **derive encryption and MAC keys** for the record layer, and to **compute the Finished message MACs** to confirm handshake integrity.

Would you like me to also show **the equations or pseudocode** for how key_block is generated from master_secret? 🔥 It's actually quite clean and interesting.

this is the transcribed lecture, the first part contains few things about kerberos , later part on ssl/tls. read the ssl/tls section carefully

system, whereby you log in once per work session, a day, for example, at work, as in a typical office, and you don't have to supply your password again. Okay, everything else happens under the covers. Okay, transparent to the human user. All right, so Kerberos does this by introducing a trusted third party, TDB, which we call Kerberos, collectively a kid distribution center. Trusted third party is a generic name, but in Kerberos, the terminology is KDC, and inside KDC, there are two components. There is a authentication server and a ticket-granting service. Now, what I did not mention last time, and maybe I did, but I don't recall, is that the authentication service knows all users. Users clients all the humans, right, all the accounts, essentially, right, but it does not need to know, it does, but it does not need to know all the servers. The Ticket-granting service needs to know all the servers, and is less concerned about users, although it has access to the same user digits.
All right, so this was the overview. User approves identity, requests a ticket, receives ticket, goes to access service. But that's a problem, right, because what we know is that it's not enough to just get a ticket for a service, right. You may not need the service right away. You may have to go to the particular service, like printing service, file service, many times a day. So that's why Kerberos separates these two tasks of, like, initial authentication that deals only with signal sign-on of the user when the user provides a password, etc., etc., and what follows later, which is user tries to access various services. Okay, let's go for that. We already did.
Blah, blah, blah. Okay, I think, do we stop here? No? Roughly? Right, so you see here in the cloud. That key distribution center houses both functions. You can think of them in two processes running on the same highly secure computer. Okay? User logs in first, gets an encrypted TGS ticket. The TGS ticket actually is for the TGS server that it will use later.
By the time it decrypts it, it has to decrypt it using knowledge of its password or password-derived key. You will go to the TGS and ask, I want to access a particular service. If the TGS decides to grant it, you will reply to the encrypted service ticket. The service ticket will then be decrypted, partially decrypted by the client, the encrypted portion that I'll explain later will be sent to the server, and then there's a brief authentication exchange

OpenAI

between the client and the server. So there's really three phases in the Kerber's sort of protocol hierarchy.
Okay, okay. So here, here I think actually what we almost got to get about that. I want you to understand this terminology. The client and the user are almost the same thing but not quite. The user is the human. The user knows the password. Okay? The user uses their laptop, smartphone,
workstation, desktop, whatever, their device, where they log in and supply the password. That device, okay, let's call it device, user device, has Kerber's client-side software. Once the user provides the password, that client-side software is client from here on. User does not perform decryption, encryption in their head, right? Software does it for them. So that's why I use the word client to denote that user-side software, which is part of the purpose. Okay, so KC is the client's key, right? Client's key, but it's not the same thing as the user password. It's a key derived from a password. Okay?
Known only to the client, to that client, and the key distribution center. Okay? Now, KTGS is a key of the TGS
for the ticket-granting service. It is known only to the TGS, so you can think of it, it shares it with nobody
but itself. And if you think, well, why are we using it like that? Because the reason is very simple. The TGS does not need to maintain or does not want to maintain a giant database of all outstanding tickets. Okay? That's a headache. The same way, the same way that on the web today, you have cookies.
Why do you have cookies? Because the web server does not want to have the headache of maintaining state information about everybody who accesses that web server. So we have cookies. So when you come back to the web server, it can recognize you, spying you, etc. But also it has beneficial side effects. Okay? So it's for the same reason that the TGS is used. Alternative would have been for the
the TGS for the TGS, sorry, the DGS ticket, the DGS is used. The alternative would be for the KDC to store all tickets locally. But then we have the headache of managing it.
Okay, so, so therefore, it uses a key that it only shares with itself to essentially encrypt for itself the fact that you have a ticket as a user. And when you come back, you better bring that ticket.
You don't have a ticket. You don't have a ticket. You don't have a ticket? Your problem. Not server setting. Or not, sorry, not TGS setting.
KV, the one I think, is a long-term strong key of each server. Printer server, file server, mail server, etc. Okay? Every server has a name V, has a KV, long-term, not converted to anything. It is a key. A real key. Okay?
Okay? Down to the server and the TGS. Okay. Next, KC TGS. That is a short-term session key that a client, C, okay, will share with the TGS when the ticket is issued.
When the client first logs in, provides the password, does the authentication exchange. At the end of that exchange, the client will learn this key.
Okay? It's issued by the AS, but it is for the client to share with the TGS because the client does not talk to the AS anymore after logging.
Okay? Puzzled? Horrified? Excited? What? Okay? Everything understood.
All right. Well, there is a shorter-term key. Okay? This is specific to a server. When a client already logged in and has this, wants to access a particular server, it goes to the TGS with its ticket-granting ticket and says,
I am so-and-so, I want to access that server, please facilitate this. The TGS will reply with a service ticket, and that service ticket will have two parts, but in those two parts, there are some redundancies,
redundancies, and this KCV is the key that is conveyed directly or indirectly to both the client and the server. So when a client is talking to the printer, or the printer driver,
the rubberized printer driver on the printer, it will use this key. Okay? That's the hierarchy of key.

Almost. There's a little more at the end.
So, this KNIT program, client, that's the client-side software. It's not the only piece of software in the client, but KNIT is the one that handles this exchange. So there is a user entering a password.
The password by the client-side software. The password by the client-side software is converted into the KC. But the message, when the user types in, in fact, username,
Alice. The KNIT does not even have to wait for her to type in the password in order to already send to the key distribution center a message that says,
hey, I'm Alice, I want to log in. And the reason is because that initial message is not authenticated. Right? It has the idea of the client,
idea of the DGS, and it seems redundant, but you'll see what it's for, and the time. The current time on the client-side. The time has to be reasonably well synchronized.
Kerberos, by default, I think, allows five-minute clocks, too. But it can be modified for installation. So the time tells this KDC, right?
Well, the AS inside the KDC, that this is recent, right? And not a replay, not stale. Okay, so then it issues, then it says, okay, do I,
the first thing it does, do I know this client? Is this client in my database? If not, no reply or error message. Okay?
If the client is in a database, the client may be not allowed to log in. In fact, Kerberos has a policy database that I will not show you,
because it's super boring, but it's necessary. It has a policy database that says, for each existing client, what is the policy for that client to log in?
A night watchman does not log in in the middle of the day. Okay? A cashier does not log in at 2 AM. These kinds of requirements, policies.
Who can log in when? A remote employee who works from home Thursday and Friday should not be logging in on Thursday and Friday. For example. Okay?
Don't quote them, just randomly generated policies. So, then, when it finds the employer, or defines the user, the client, and it finds that the user is allowed to log in,
it will issue this TGS ticket, or TGT, Ticket Granting Ticket. It's called Ticket Granting Ticket because it's not really a ticket in and of itself, it's a ticket to other tickets.
All right. So, client obviously cannot forge that ticket. You'll see why. Client can, of course, if the client is a real user and knows the passwords and can compute KC,
he can forge part of that ticket. But what he cannot do is forge something for the TGT. Sorry, for the TGS. Because he doesn't know the key of the TGS.
What you see here in green is an encryption. It's essentially a two-layer encryption, but the outer layer is encrypted under the KC, the client's password-derived key.
And inside, there is a blue KC TGS, right? This is the session key for this entire session, for the, until the client logs out, or the TGT expires, whichever comes first.
Because you see there is also ID TGS, it better match what the client says. So when the client decrypts this, first of all, if the client cannot decrypt this, like he gets garbage, that means two things.
Either the client is lying, meaning the user is not the right user, or the user mistyped the password, it happens, or somebody modified this green blob on the way between the KPC and the user.
Totally possible. Could be innocuous, like a router is buggy, or could be malicious.
Okay, so failure to decrypt can be any of those things. If the decryption succeeds, the client will make sure that ID TGS he sent in the blue is the same one that comes back in the green.
It will also make sure that time KPC is reasonably near the time C that he sent earlier.
And the reason is because time C is in fact acting as a challenge. Yeah? You see that? Time kicks up. So it's kind of acting as a challenge.
Lifetime.
Lifetime. Clearly, that's the ticket lifetime for that, for that, for that ticket granting ticket. After that lifetime, you have to re-login.

And then the last, but definitely in all this, that's a big part, that red part is ticket TGS. That is encrypted using TGS's key.
Client cannot read it. It just treats it as a block. But if the client decrypts, and everybody, by the way, trusts the KPC.
KPC is called a trusted third party for a reason. You have no choice but to trust it. It's your big brother.
If the client succeeds in decrypting the green part, right, the outer encryption, it knows the structure of the red part. It just cannot read it.
But it knows its structure. It knows what should be inside. And inside, among other things, should be an exact copy of that blue KC TGS.
And better be the same lifetime, too. Otherwise, the curve of simplification isn't correct.
Right?
That key is freshly generated, randomly generated by the AS.
Right? And so what's inside? Roughly here, this is not an exact format. What is inside is that same blue P, as I said, the ID of the client, for good reason.
Address of the client. Actually, it could be like a, depending on the version of the purpose, it's either an address, IP address, or a list of IP addresses.
Okay? Or it could be like blank or like asterisk, which means any IP address. Not advised.
ID TGS.
ID TGS.
ID TGS.
Well, that's what TGS is going to allow, okay. It's neat.
Time.
KDC's time, which is the same as the time in the green.
And lifetime, which is, again, the same as in the green.
So there's a lot of duplicate information, and it has to agree, because that's what KDC always does.
It makes sure that whatever is inside the red part is faithfully replicated inside the green part, okay?
In fact, there's nothing really secret there. You see, the client can know what's inside.
But he just cannot decrypt. So, client decrypts with, like I said, green part, the planes keeps TGS, gets the six tracks, the ticket TGS, bam.
Now, it goes to the TGS, okay, with this TGT on, this ticket from before.
But it only goes there if it needs to.
If the client came to work, a user came to work, used the workstation to log in, and wants to, like I said before, take a break, no problem.
He just wants to run something locally that requires no server access, no problem.
Kerberis is not involved. The moment the client wants to use some server that is Kerberized, right?
It's protected by Kerberis, like print something, alright?
Or use some CAD package, or access a database, or whatever.
So what it does is it sends the following thing.
It encrypts, under the case TGS, which he just extracted, or earlier, not just, earlier extracted from the previous packet,
to encrypt its identity, right, its address, where it's coming from, and the current time.
And that becomes authentication token.
Okay?
Off C.
In addition to that, it adds the ID of the service.
It wants server, he wants to access ITV.
And that red ticket TGS from before, which he cannot read.
But we remember what's inside there, right?
So he sends it to the TGS, and the TGS does what?
It looks at ID and says, huh, well, actually, I'm missing something.

There's, you'll see it when I give you a full format.
There's, also, identity of the client is sent in that message.
It's kind of implied.
So the, the ticket granting service says, okay, client Alice wants to access server printering.
So it's, is that allowed?
So now there's a policy database, right?
There's, like, a user, remember I said there's, like, a user database that lists users and when they can log in
and all kinds of policy about when can they, can they get onto the system.
And then there's a separate policy database that says, which is essentially an access control,
a representation that says, does this user have the right to access this resource, this server, at this time?
Okay?
That is called access control.
Okay?
So, if that fails, the TGS doesn't bother.
It just returns an error.
Access denied.
Maybe it was, I forget, maybe it was some code.
The next thing it does, it authenticates, well, sorry, the next thing it does, it decrypts that red ticket.
And because the red ticket is encrypted under its own key, it can't decrypt it.
If it cannot, then, again, either the client is supplying something wrong, or somebody in transit modified it.
Right?
You have to always consider the possibility.
But if it manages to decrypt that red or orange part, then inside it, you will find what we already saw.
Inside it, you will find this.
This, this, this, this, this.
Okay?
So, among other things, you see the first, the blue part is the KCTGS.
He will extract it.
Right?
He will extract it, and then he will verify off C.
Off C is computed with KCTGS.
That's the key she would extract.
If the off C does not verify, again, two things are possible.
Either the ticket has been hijacked, and somebody is, or the client did something wrong, or somebody modified it in transit.
For the message to be fully processed, the TGS has to be able to decrypt the ticket TGS and verify successfully off C.
And, of course, the policy part, but can this client access this service?
If all that succeeds, the TGS replies, okay, with, again, double encryption.
That's a standard format inside, inside Kerberos, double encryption.
Today, it doesn't do that, but this is canonical Kerberos.
KCTGS encryption, which the client knows, or should know, right?
KCV is the new key, randomly, freshly generated.
That will be the key for the client to talk to that print server.
Not yet, right, but to be.
IDV, meaning the server ID.
Time on the TGS side, and the ticket V.

The ticket V, in whatever color that is, beige, is not, again, not decryptable by the client.
But the client pretty much knows what's inside.
You see, similar to what we saw on the previous slide.
It's an encryption on the server V's key, long term, strong key.
On the same KCV, ID of the client, address of the client, ID of the server, just for sanity.
Time on the TGS side, and the lifetime of the ticket.
This is how long can this user, Alice, use this particular printer,
without having to go back to the TGS and get another ticket.
This lifetime cannot be longer than the TGS ticket lifetime.
Make sense?
Otherwise, it would be really stupid.
Right?
You don't want to give somebody a right to say print past the time they should be logged back.
Or, for example, access an employee database.
Okay.
So, the client does the same thing to do with TGS before.
It will, because it knows KC TGS, it will decrypt that outer part, obtain KCV, obtain IDV.
It will compare IDV to the one it sent in the first message.
They better match.
If they don't match, there's something wrong.
It will check the time TGS to make sure it's a recent time.
And, you cannot verify TKD, but it just caches it.
Okay?
Caches it because we need it again.
Okay?
Everybody gets it?
And now it's time to go obtain CERNs.
Okay.
So, you want to actually print.
Okay.
So, now you have the client.
Actually, what I said, the user just says, I want to print.
But all of this other stuff, like going on the previous slide and this slide, that happens under coverage.
The user, human, is not involved.
Right?
Unless there is an error, right?
Like access denied or some sort.
User is not bothered with anything.
It all happens quickly.
So, the user, client, excuse me, goes to the server directly now, finally, and says,
Yo, I have a ticket for you.
Okay?
Here is the ticket.
That pink thing.
Came from the previous slide.
Right?
Here is an authenticator.
And here is an authenticator.
To prove that I know the same key that is included in the ticket.
Why?

Well, because if I don't know the key, then it's not my ticket.
Or I'm not me.
So, the server receives that ticket.
The server, first, decrypts that ticket using KD.
Right?
Decrypts the ticket.
And inside finds what we already saw.
That pink thing.
And it finds KCD, IDC.
Right?
It will make sure that the ID of the client is the same one that the message of the next slide comes from.
The address.
The IP address should be the same.
Although those are fungible.
We all know this.
It will send it to check its own ID.
Does my ID match what is inside the pink thing?
The time TGS has to be recent.
Right?
But the important thing is the lifetime.
Okay.
Okay.
Now, the interesting thing is time TGS has to be, does not have to be synchronized here.
Does anybody see why that is?
All the previous times, when you, all the previous times you saw time, it was like they have to be reasonably linear.
This time TGS does not have to be reasonably linear.
This has to be within like time.
Exactly.
So, if a printer server receives a ticket, a service ticket from some user, and the inside when it decrypts the ticket, it's part of the ticket, it says time TGS 8 AM, but it's now noon, that's okay.
That's okay.
Because the lifetime, right, as long as the lifetime is okay.
Because it just gives it time TGS plus the lifetime, that's when the ticket expires.
Okay.
So now, we're almost done.
All right?
So the server extracts, extracts, decrypts the pink ticket, extracts the information, and then verifies off C. Why?
Because the same Kcb better be inside there that was computed with, off C was computed with, checks the idea of decline, again, the rest of the time, fine, that time is better be timed.
That off, right, that off C better include the time that's, like, near.
Synchronized.
Not perfectly synchronized, but somewhat synchronized.
Okay.
Last message.
Strictly speaking, it's an optional message.
You can configure curbers to just do one way authentication, like this, to client to

the server.
And the reason is you might want to send a print job to, let's say, to the printer, and
you don't care if you hear anything back.
Right?
So the client can just take that, what is that color, lilac, that first message in lilac,
and append a print job to it, like a PDF file, and say, here, let's print this one.
So that last message might not always be necessary.
But typically, I think, by default, it is.
So it's an encryption on the KCV of the time, time C is from client off C, right?
So it's kind of a challenge, right?
Time C functions as a challenge.
It proves to the client that the server received the previous message, right?
And, of course, he knows KCV.
Otherwise, nobody would compute that time C plus 1.
That is actually one of the vulnerabilities of this version of Curveverse.
But that's for another lecture.
Anyway, for now, let's consider this reasonably secure.
Right?
That's the idea.
Right?
So you have essentially, for a given user who starts out fresh, comes into work, or just
logs in for the first time, there are three exchanges.
One is to log in initially.
The next one is, every server you want to access, you have to get a ticket for that server.
And then the third is, you do a handshake exchange with that server.
Now, if the client has multiple jobs to print, let's say, the client gets the tickets for
printer server at 9am, and then it prints something.
And then at 10am, the client has something else to print.
As long as the lifetime of this ticket is valid, the client does not need to go back to the DGS.
Right?
It can reuse the same ticket.
And that's why we have the lifetime.
Okay?
Yes?
Clear?
So, let's just go through the reasoning.
For the last, I forgot to mention this.
Right?
So, the server, let's look at it from the client perspective.
The server can only produce this message if he knows the KCB.
Because only the server.
Client does not generate messages like that.
Like this new message.
Right?
Because the client is not a server.
It is not programmed to generate such messages.
So, the only party that could generate such messages is the right server.
Because it is the only other party.
Well, except for the trusted third party who we trust not to do and interfere.
So, only that server can do.
Server, on his part, the server can only learn the KCB if he could have decrypted the ticket.

There's no way he could learn the KCB without decrypting the pink ticket.
Okay?
And verifying everything.
So, everything was A-OK.
Alright?
And the server can only decrypt that ticket if he knows KB.
Right?
Because that ticket, that pink one, was encrypted with KB.
Well, then, if the server knows KB, then it must be the right server.
So, therefore, it's authenticated.
That's kind of the change of logic here.
Here's the, sometimes the bird's eye view comes again.
Okay?
So, here's the high level of Rubio Athenians for the Stallings book.
So, it's probably a different terminology but it may help you understand that.
So, again, once per user login session, you have the request ticket, granting ticket, which is essentially login with authentication server.
You see that all this blob is one machine running two processes or two functions.
You get a ticket and a session key.
Then, when you want to use a server, you request service granting ticket, service ticket.
If successful, you get back a ticket and a session key.
And then, when you actually use the service, you request service with that server.
And, if it succeeds in authentication, it will return a mutually authentication.
Okay?
And, in another way, more specific.
These are the messages in Kerberos.
One day, if you work in the security industry or even for security, doing security function in a non-security company, you might have to deal with this.
Because a lot of industry uses Kerberos in one way.
Sometimes it's called Kerberos.
It's integrated like in Windows Enterprise or some other larger software packages.
But, underneath it is Kerberos.
So, again, client, server, KDC.
First message is called ASRequest.
Authentication, service request.
That's the login allowed by me.
The reply with the TGT is ASReply because it comes from the AS file of the KDC.
Next, when you want to use a service called TGSRequest, request from the ticket granting service.
The reply, if always good, is TGSReplied.
Then, finally, app.
Application server request and application server reply.
Ad nauseam added to that.
And here's the gory detail of what's inside.
Okay.
Now, the terminology here is a little different because it's formal.
Right?
Formal from the spec.
So, ASRequest, principle C is the name of the user.
You can think of it as the client's name.
The next one is around, and I'll jump ahead and say that Kerberos works for a single administrative

entity.
Meaning that you cannot have several organizations living under one Kerberos umbrella.
That doesn't work.
Each has its own.
If you have different companies, different parts of the same company, you may want to have two or different multiple Kerberos installations.
I'll say a few words by the way.
But this is the, each Kerberos installation is called a round.
And you need to specify.
Like, Kerberos is reserved.
It is, it's a, it's a request to Kerberos, for Kerberos ticket granting ticket.
And then the uppercase realm, that's configurable.
Right?
That's whatever the name of your realm is.
IP list is the list for potentially empty of IP addresses for which you are requesting a TGT.
Now, the policy may disallow this.
And only allow one.
But, policy allows it to specify multiple.
Say, for example, I'm going to use an IP address of my MacBook and my desktop and my smartphone.
Okay.
But, if the IP list is empty, that means I want a TGT that's independent of the IP address.
I want it to be portable.
Again, that might not be allowed.
That depends on the installation.
Lifetime.
I want the ticket for this lifetime.
That may not, may or may not agree with the lifetime the, the, the, the, the AS wants to issue a ticket for.
Anyway.
Okay.
That's AS request.
AS reply.
Let's stick to that, let's stick to that second line.
AS reply is the same, the realm.
Right?
Just echo it.
Okay?
Timestamp on AS.
Lifetime, which cannot be longer than what the user asked for.
Can be shorter.
So, the user can ask for 24 hours but only got 8 hours.
That's okay.
Lifetime, and KC, TGS.
Right?
So, that's the key.
Remember, we already talked about it.
And all that curly brackets, curly brackets denote encryption.
All that is encrypted under KC, which is the client's password or IP.
And then, that little seeming little part in yellow is TGT and encrypted for TGS.
Notice what's inside that.

Okay?
It's encrypted, right?
So, I only show you the unencrypted part on second line.
It has the name of the principle, which is the same, the name of the user, the round, the IP list, the timestamp of the AS, lifetime, and KC, TGS.
So, it actually duplicates a lot of the fields you will see on the left side.
So, it's encrypted.
So, it's encrypted.
So, it's encrypted.
So, it's encrypted.
So, it's encrypted.
So, it's encrypted.
So, it's encrypted.
And the KC, which is encrypted.
And the KC, which is the client's password or IP.
And then, that little seeming little part in yellow is TGT and encrypted for TGS.
Notice what's inside that.
Okay?
It's encrypted, right?
It actually duplicates a lot of the fields you will see online.
Great.
All right.
And TGS request.
This is when this client goes to the TGS and says, I want a ticket for the printer.
That contains principle V.
Principle V is the name, official name of the printer service or whatever service you want access.
Lifetime.
For how long?
Again, cannot be longer than the lifetime of your TGS ticket.
And an authenticator, right?
You don't come to the door without an authenticator.
So, the authenticator one-off-one is an encryption of the principal C, meaning his own name, timestamp,
current timestamp on C, the client, and checksides.
Okay?
It's an authenticated checksides.
Very simple checksides.
Okay?
So, TGS request arrives.
It is decrypted and verified.
Okay?
And when it's successful, everything goes well.
The TGS replies with this, with the service ticket.
And it contains two parts.
Remember?
The name of the server.
The timestamp.
The timestamp.
The timestamp.
The timestamp.

The timestamp.
The timestamp.
The timestamp.
The timestamp.
The timestamp.
And it contains two parts.
Remember?
The name of the server.
The timestamp.
The timestamp.
The timestamp.
The timestamp.
The timestamp.
The timestamp.
And theuler of the przyst.
As for the client.
So, this part, this part is decrypted by the client.
And then the next part is only decrypted by the server, when the client decides to go to the server.
Okay, then these are where the last two messages when the client goes to the printer server and supplies an authenticator
and then takes this from here, just per meter, copies it here.
Okay? And yeah, this is time sample, 400 plus 1.
Any other questions? Any questions at all?
It's not as difficult as it might seem. The notation might be a little obscure, but stare at it for a while, and then for a while, everything will be clear.
No question. Okay.
Now, as I said before, Kerberos isn't made for large networks or administratively heterogeneous environment.
So, you want to divide even if you have a large company, even if it is one administration, but you have different geographical locations
and just very large installations, like maybe you have one building, but there are really different departments.
If you have a product department and a sales department and marketing, maybe you want to separate and have two different Kerberos realms for those.
It's just easier to do like that. But every realm has to have a KDC. And every realm has to register its own server and its own users, right?
So, now, how do you interoperate? Well, that's done in a kind of very obvious or naive way.
And so, one way to think about it is, like, imagine you are at UCI, and we run our Kerberos realm, which we actually do.
And then we have, let's say, UC Riverside, about 50 miles away. It's another UC campus. They run their own.
So, they'll have ROM at, you know, Kerberos at UCR, and we have Kerberos at UCI. Those are two different realms.
But because we are all good UC citizens, we are allowed to access each other's resources.
Okay? So, how do we do that? Well, you want to do that, you first need to log in locally, where you are known.
Right? So, your UCI, your UCI denizen of sorts, right? You log in here, you get access, and then, you go to the TGT,
to TGS, and you ask for a special TGT. For a special ticket. Not a TGT. So, you follow me, you log in, you get the ticket-granting ticket.
And now you go to the TGS and say, I want a ticket for a server, but it's a special server, it's called

KDC at UC Riverside.
So, the way that it's treated is like another server, but it has to be explicitly registered locally.
And it has to be bought. It doesn't have to be bilateral, but it makes sense if it was bilateral.
Does that make sense? But you're not done. The process is more complicated than that.
Because once you get a service ticket for a TGT, for a TGS in Riverside, you then have to, you don't need to log in.
You don't need to log in. But you need to bring that service ticket and contact directly the KDC at Riverside.
And it will say, oh, this is a foreign issue ticket, let me look in the database if I know of UCI KDCs.
Oh, yes I do, in which case, you know, you can get an application ticket on me locally at UC Riverside,
and then you can get, you know, printing access and file access and all that. Okay?
So, it's not, like, trivial, right, in terms of, like, what's, the particles are trivial, given what you already saw.
It's not really very different, but there are more of them required in order to support cross-ground functionality.
And also, every KDC, then, has to share a key with every other KDC.
So, if we have 10 UC campuses, right, and each UC campus runs, you know, its own Kerberos installation,
there have to be 90 keys, right? And, you know, every UC campus has to share a key, right?
A KDC, you know, every two KDCs have to share a key.
And that has to be done manually.
Now, one of the important things to take away from Kerberos is that it's not super scalable,
it's not like using public key, you know, in the world of web browsing, okay,
where things scale to the entire world with questionable security, of course.
But we feel like they scale to the entire world.
And Kerberos things don't scale that far.
But they scale to reasonable size, medium-sized organization.
And that's important, okay?
The other interesting thing is Kerberos is basically a symmetric key.
It does not use, by default, right?
It's in its vanilla form.
It does not use public key cryptography at all.
Which, for example, if you know anything about what's going on in the world of tech today,
of science, makes it kind of attractive for our quantum resistance.
Because quantum technology will, sooner or later, make most public key techniques weak or obsolete.
And that would leave us with either some very exotic public key techniques,
of this security which we're still not sure about,
or it would leave us essentially with symmetric key work.
Like what Kerberos does.
Anyway, so that's the detour.
So that's a nice feature.
The services that use these short-term session keys.
For both, like, client login session.
The session key lasts for as long as the TGT is valid.
And even short-term client application key, right?
Meaning client-server keys that are used per service.
Now, current release, I think there was a version last year,
right middle of last year was released.
This is 1.13.

You can, Kerberos is fully public domain.
If I had a TA and this was a larger class, I may have,
I would have had maybe some actual exercises with Kerberos.
Because you can totally install it and play with it.
It's not that difficult.
It's reasonably well maintained and it has good documentation.
In order to use it, you have to, of course,
install the client-side software.
Not only that, but the application service.
So if you want to Kerberize your printing,
you want to Kerberize a certain database access,
you have to essentially reinforce every service, right?
So it usually requires a bit of work.
Not a huge amount of work,
but you have to link Kerberos server-side libraries.
And so you cannot just take a software,
generally I don't think you can take a software package
as it is and just plunk it into Kerberos shell
and say it's Kerberized.
It has to be Kerberized, Kerberos-friendly.
It may come Kerberized already from the manufacturer,
from the software provider.
That may be.
But if it doesn't, you have to do it yourself.
Right.
So the current version of Kerberized,
like I said, it's released 1.3.
Version is called 5.
You can't see it over there.
It's obscured by the zoom logger.
But the version I described to you is more older
and still in use.
It's called version 4.
This is where version 5 has certain interesting modifications.
For once, it addresses one of the problems in Kerberos,
which is the first message,
the first login attempt message, right,
by the user is in clear text.
You recall that?
The key, the password derived key is not used in the first message,
from the user to the AS.
And what that says is that if you're an adversary
and you know the name of the user,
then all you need to do is send a clear text message,
pretending to be that user.
And what you will get back is a nice present.
And the nice present is, well, it's an AS reply.
Remember, that's that message.
But remember that nice present carries in it,
AS reply carries in it this first part,
which is encrypted in the KC.

Yeah?
That's a password derived key.
So if you can brute force the password,
you can decrypt it.
Yeah?
Oh.
No.
You just brute force.
Basically, you just brute force the password, right?
It gives you something to play with.
It's essentially like you just got an entry
from an Etsy password file.
You can just try to decrypt it
under every possible password derived key
until you find inside the name of the principal, right?
You'll know when you hit pay dirt.
You'll know when you guessed the key correctly.
Right?
So it's subject to dictionary attacks.
The alternative is very, very simple.
You require what's called pre-authentication,
which means you require the client to prove knowledge
of the password or IP in the first message.
And that's something that Kerber's 5 supports.
That's called pre-authentication in the initial
answer request.
That's not without problems itself.
Yes?
The attacker receives that pre-authentication.
Exactly.
Exactly.
Well, let's talk about it for a second.
Does everybody understand what he just said?
Are you following?
No?
No?
Hello?
Sign up with your phone?
Laptop?
Big thoughts?
Worry about the economy?
No?
Does everybody understand what he just said?
Yes.
If you put, if you require the client to authenticate itself
in the first message, then there will be something
obviously computed with a password-derived key
that is present in the first message.
That means, yep, okay, that's an opportunity for a password attack,
right?
A brute force attack on a password or dictionary attack.

But are they the same?
No.
The main difference is you can't just get an arbitrary
You get, see, in the first case, right?
With the current first version I showed you, you get something
for nothing.
Right?
You get something for nothing.
You just generate a clear text, Alice, blah, blah, blah,
timestamp, how difficult to get the correct time, et cetera.
And you get back to present.
With pre-authentication, indeed, you are still subject to dictionary attack,
but you got to be lucky and wait, you know, lie and wait, right?
Lie and wait until the victim user logs in.
You see what I mean?
So it's not like, you know, oh, at your convenient time,
at 3 a.m., we're going to generate a bunch of these
and get back your class.
No, no, no.
You have to be there when Alice comes in, you know,
and let's move that.
But, so there is a difference.
It's not a huge difference, but it is a difference.
Right.
So the other modification in Curves version 5, which is today's version,
is this in client-server application.
What I showed you before, the client and the server, application server,
use the same KCV.
Remember KCV?
Like, you wanted to print, you used KCV, right?
Or, you wanted to print again using the same ticket, you used again KCV.
Well, that's considered poor cryptographic hygiene.
To reuse the same key for different, like, let's say, server, client service sessions.
So instead, Converse 5 derives, uses the KCV as a master key for the ticket,
and then derives individual subkeys for each, like, short client-server session.
Does that make sense?
Shorter, shorter, shorter.
Yeah, that's right.
So you got your ticket to use the printer for the next 5 hours.
During those 5 hours, let's say you printed 5 times.
In Curves version 4, you would have used the same KCV to print.
In Curves version 5, you would use 5 different subkeys.
Okay?
Now, it should be abundantly clear to you, right,
that the way that the servers are treated in Curves version is as being stateless.
Does everybody understand what I'm saying stateless?
The same way that we don't want the TGS to keep track of the issued tickets
for all the currently loading users, because it makes it heavyweight and unpleasant.
Even so, even more so, it makes it unpleasant for the servers to keep track of tickets.
Because imagine what would happen.
All the users come in the morning, log in, and start printing, and accessing databases,

so all the poor servers would have to remember all the tickets that the users brought to them, right?
Cache them.
That's overhead.
You can lose things, isn't it?
But instead, Curves is stateless.
The server in Curves does not know any users.
Does not know you from you from me.
No idea.
A Curves server only knows one thing.
Its master.
And its master is the KDCs.
Or, more specifically, the Ticket Granting Service.
And it shares only one key, long term, with that Ticket Granting Service, right?
So whatever the master says, it obeys.
Right?
And keeps no state.
Keeps no state.
Which is very nice.
Because otherwise imagine, you'll print, like a printer is a good example.
Because printers are usually zanemic devices.
Right?
They're good at printing, they have fast mechanical parts and whatever.
But, in terms of computing power, they're not very powerful.
And they don't have a lot of storage.
Because in some ways, printers are like embedded devices.
So, you don't want to overload them with having to cash tickets.
So, it's good.
Now, the other thing is, what you can do in Carver's D5 is you can do what's called forwarding.
Take that forwarding.
Yeah?
How much compute do the heavy duty printers have?
Well, you know, if you pay $50,000 for a 3D printer, that one will definitely have compute power.
But even there, it's such a precious resource.
You don't want to overwhelm it with like caching tickets.
Why?
Right?
I don't know.
Well, it's like the ones we have in DBH.
Yeah.
Those are like connected to the internet and everything.
They are.
Right.
That's going to be a lot more power than my printer.
No, for sure it will have more power.
But it's still not a full-blown computer, right?
It will probably have, my guess is that something that we have in DBH will be a power of an iPad of some sort.
Maybe a little better than an iPad, you know?
Some kind of embedded like, like a, like a airplane entertainment system type, you know, device.

Or a car, infotainment system in a car.
But not, not a lot stronger than that.
But also, it's not just compute, it's also storage, right?
That's the other thing.
If you have a cache, it better being super fast and expensive memory.
Yeah.
So, but also recall, Kerberos was designed, did I say when it was designed?
1987.
Okay?
Your parents were in kindergarten.
So, the internet already existed.
But, you know, it wasn't like this, like it is today.
But, so, considering that something like Kerberos aged relatively well, I would say, it's still being used.
And it still kind of resembles its original, it's not exactly the same, but it looks very much like what was there in the early 90s at least.
That's a long time ago.
There are very few products you can say this about.
Right, so, authentication forwarding means that you can forward the ticket.
Like for example, if you're, if you're accessing a file server, it needs to access, and it needs to access your box.
If you're accessing a mail server, it needs to access a file server.
So, in the original Kerberos, you would have to get two separate tickets.
And it was clumsy.
And now, the ticket that the file server gets can be forwarded to the mail server.
And it can honor it.
So, there's like some games to give you from that.
There are, the Kerberos 5 supports hierarchies of realms.
So, what I described to you about inter-realm thing and splitting things in Kerberos realms.
Kerberos V5, if I'm not mistaken, I think I'm not, supports hierarchies.
Where you don't have just like multiple realms as peers, but you have some super realms.
Alright, and there's other stuff.
Oh, it also supports multiple description schemes.
The original Kerberos used the DES with some weird mode of operation that was not standard.
And there were some problems because of it.
Now, you can configure Kerberos to use pretty much any decent encryption function in the hash function or MAC function.
But just to give you an idea about pre-authentication, coming back to that feature.
So, as I said, anyone in Kerberos version 4, the way you, the one you sort of described, can use the initial packet as a vehicle for obtaining, you know, gifts.
Right?
Things encrypted with passwords and so on.
So, it makes password guessing easy.
In Kerberos V5, the first message, the AS request message that you saw earlier, requires another field called PA data.
And basically, it's like, you can think of it as a timestamp, client's timestamp, encrypted under the client's key.
It just says, look, this is my timestamp, my current time, encrypted under my key, presumably only I know the key.
Well, the AS.
So, the AS only replies if it can be encrypted.

Okay?
But in reality, it's done more like this.
Typical configuration is, the client will send an AS request, normal one, without any authentication.
And the server will reply, the AS will reply with like a special error code.
It's an error.
Pre-authentication required.
Don't come back to me until you demonstrate the knowledge of the KC.
And so, that's number three is AS request with the PA data that I showed you before.
And now, AS is happy if it issues a tissue.
Everything else is the same.
Come on.
So, re-authentication is one new thing, but there are extensions.
For example, today, Kerberos can be 45 with something called pkinit, which, if I remember correctly, allows a user to securely fetch his public key.
So, the user can be registered with a public key.
But the user does not need to remember their public key.
So, the idea is that the user tries to walk in.
The user's public key will come back and allow, sorry, private key.
The private key will come back from the AS and will allow the user to use that private key for all subsequent exchanges, so that Kerberos is fortifying it.
The other extension is for public key based cross domain authentication, not important.
So, what is Kerberos used for?
For many things, like I said, Windows Enterprise Networking uses it.
And if you've used Windows on an enterprise scale, even like here in ICS, you probably used Kerberos without knowing it.
You may have occasionally seen some errors that will tell you, that will leak information that you are using Kerberos.
Anybody seen those?
Anything starting with KRB?
All right.
So, it's also used for securing, if you are a Unix person, and you remember these Unix commands, like remote shell, remote exec, RCP remote copy.
Right?
So, these are insecure commands.
Right?
Natively insecure commands.
FTP, right?
If you secure them, you can do so, if you want to secure them, you can do so with Kerberos.
And then it becomes essentially Kerberized, RCP, Kerberized, RXAC, Kerberized shell, etc.
You can also Kerberize the superuser command.
Yeah, that's, Jesus.
Let's see quickly what we're going to look at next.
Okay.
So, that concludes Kerberos, and next we're going to go and look at web security.
Oh.
Well.
SSL TLS.
Any questions about Kerberos at all?
Okay.
Okay.
So, everybody knows SSL.

Everybody knows SSL.
TLS is kind of the same, sort of, kind of.
Anybody know what they stand for?
Ask them.
Ask them.
Ask them.
Google.
Or your favorite chat to be.
Secure sockets layer.
That's SSL.
Transactional layer security.
That's TLS.
And SSH is secure shell.
Okay.
Okay.
So, we're going to go through the part.
The particle is complicated.
But I figure if you know anything, if you want to know anything about today's security landscape in the real world, you do need to know about Kerberos.
You do need to know about SSL TLS.
And you probably need to know about something called IPsec.
Does anybody know what that is?
Okay.
Two people.
Okay.
That's IP security.
To be covered later.
So, these are all sort of standards of sorts.
Kerberos is actually a standard.
Does anybody know what internet standard means?
Does anybody know what IETF is?
Very few people.
How do you live in computer science world and not know this?
IETF is Internet Engineering Task Force.
It's an organization run by volunteers that essentially governs how the internet works.
Everything about the internet.
Well, not the web, essentially.
The web is a web consortium.
But the internet, essentially, and just above the transport layer, is governed by ATF.
Internet Engineering Task Force.
It's an international organization.
They have working groups.
And these working groups are concerned with everything from, like, network layer.
I mean, they don't work on Wi-Fi.
So, they don't work with data link layer.
Because that's not interesting.
They don't work on radio jamming or physical layer.
Radars and microwaves.
No.
IP and IP.
Right?

Actually, between IP and, let's say, a session layer.
Everything there is subject to IETF approval.
And they publish official documents that are called RFCs.
Funny enough, it stands for request for comments.
But it's not really request for comments.
It's usually an internet standard.
Okay?
So, Kerberos has an internet standard.
If you go look, RFC, Kerberos, you'll find it.
It tells you exactly how Kerberos works.
If you look at IP, V6, for example, it has V4.
They all have RFCs.
And they tell you exactly how IP works.
Which means that if you follow the standard, you can implement that protocol, that format, in whatever language you want, in whatever platform you want.
From a smart toaster to a supercomputer.
And they will be able to interoperate.
If you implement correctly.
Okay?
So, Kerberos is a standard.
So is SSL TLS.
And so is that inside.
All right.
So, history.
SSL goes back to the 90s.
I remember version 2.
It was like late 90s.
Version 3 is what we kind of use today.
Right?
Because version 3 is essentially TLS.
Don't ask me why they switched the names from SSL to TLS.
It had something to do with trademark dispute.
Okay?
But there is an RFC 22.6.
So if you just search on RFC 22.46, you will find a very detailed specification of SSL.
Sorry, TLS.
And if you want to look at the open sources limitation, there are many.
The most popular one is this, open SSL.
Okay?
It also implements a lot of crypto stuff.
So if you're ever dealing with crypto stuff, and you want to dig, and you should not be baking anything on your own, because you will mess up royally.
You should use that.
Because it's been around for a long time.
And although occasionally there are bugs, there are fewer bugs in that library than there are in most others.
So, SSL TLS, you secure it in the transport layer.
It sits just above TCP.
Okay?
Although these days, there are also to sell versions for UDP, but we will not do something else.

That's a separate topic.
So, think about TCP.
Everybody knows transaction control protocol, transmission control protocol, depending on who you ask.
On top of it, it's going to be TLSSL.
It's optional.
It's not always present.
Okay?
But it's secure socket, right?
That's where on top of TCP you have sockets.
Okay?
If you want to TLSSize your applications, like with parameters, you need to change them.
Okay?
Now it says here, because originally it is worth my slides, that it's a thin layer between TCP and HTTP.
That's not actually true.
It's not just for HTTP.
Yes, most things that use, or the biggest user of TLS is the HTTP protocol.
Okay?
So whenever you have HTTPS, right, and you see HTTPS in your browser window, you are using SSL.
Or if you use some godforsaken browser, maybe they'll have like some little lock symbol, like they used two years ago, instead of HTTPS.
So that might not be spelled out.
But if you see a lock symbol or HTTPS explicitly, of course, you are using TLSS.
It comes with support, right?
So most reputable browsers, clients, and of course, HTTP servers come with SSL TLS support.
So it's built in.
There are two layers in the TLS SSL.
The layers should not be confused with particles.
Okay?
There is called the record layer for just transmitting packets.
Okay?
And the upper layer where you do what's called protocols, right?
And basically record layer is just treating one SSL segment or packet at a time.
And at the upper layer, you do all this kind of management.
You will see what I mean in a second.
So it kind of looks like this.
Okay?
So you have this TCP and you have SSL record protocol.
And then on top, you have these other protocols.
Okay?
So it looks weird, I understand.
Because HTTP, you see HTTP is shown as being on top of the record layer.
And SSL itself, as the N-shaped protocol, is shown at the same layer.
Well, it doesn't make sense yet, but it will.
So you just read this.
My throat is getting raspy.
In some ways, you will see some of the same concepts as you saw in Termverse, but SSL TLS is not a symmetric, sorry, it's not using symmetric cryptography throughout.
It uses, like many other products or software, it uses public cryptography initially to establish, to

help establish the secure connection between the client browser.
I say that because that's the biggest application.
But keep in mind, SSL TLS is not just for browsing.
Okay?
But I will use browsing as an episode between your client browser and the web server on the other end of the world.
All right?
It's using public cryptography to establish, just in the very beginning, it minimizes, in fact, the use of public cryptography, but it does not avoid it.
Okay?
And once the initial connection is established, public cryptography is no longer used.
All right?
So, but there's no trusted third party.
Okay?
No KVC, no AS, no TGS.
There's just a big, bad world out there.
Okay?
Where innocent little clients talk to potentially dangerous servers.
All right.
Let's look at the actual protocol.
Okay.
So, we're going to use terminology client and server.
So, I know it's a little bit confusing, but it's actually not because if you think in Kerberos, client and server, where?
Well, user and client and server.
So, here are two.
We just don't have KDC, AS, TGS anymore.
Let's see.
In TLS SSL, 99.9% of the time, only the server is authenticated.
Okay?
So, that's the philosophy also that is different from Kerberos.
Only the web server, right?
Think of the web.
Only the web server is authenticated.
Have you ever, has anybody ever seen mutual authentication in CSL TLS?
Assuming you know what I mean.
No?
I've seen it twice, three times in the last one year.
So, TLS does support mutual authentication, but generally does not do it because it actually has little justification.
And the reason is like this.
When you are sitting in front of your browser and you type in or paste a URL, think about what happens, right?
Something gets parsed by your request.
And there's like three parts really to, well, four parts to the URL.
There's the protocol identifier, which is the first part, right?
Doesn't have to be HTTP, by the way.
But usually it is, right?
SSL is HTTP, yes.
Then there's like delimiters.
Then there's something called the domain name.

Right?
It's actually an address, sorry, a host name or a name that can be used to resolve to, you know, be resolved by DNS, right?
Domain name service to produce an IP address.
So it actually refers to an entity.
And what follows it?
There's a slash what?
What happens after www.uci.edu slash what?
Maybe nothing.
Maybe nothing, right?
Well, wait.
The domain name is where you want to go.
What's after that?
What is it called?
Path.
Path, yes.
It's the path.
It looks like a Unix path, right?
It might even be.
It doesn't have to be.
Could be interpreted differently, but could be a Unix path, right?
Like go to that machine, go to that web server, that physical host, and go slash, et cetera, slash password, blah, okay?
And then what?
And then there might be another component after that.
Yeah.
Frequently overloaded.
That's why you see these URLs that spread for miles.
Right, so what are we concerned with when we browse the web?
Fishing?
Squatting?
Anybody know about squatting?
Mistyping?
You mistyped one letter or one number in the URL and suddenly you are in a God knows where?
Might pretend?
Yeah, you type wellsbargo.com and you might wind up in a site that looks like Wells Fargo, but it ain't.
And here you are entering your username and password and your account is toast within minutes.
Without SSL TLS, it would be.
So, what's important for you, the user, for us all, is to go to the right website, right?
When we click on the bookmarks, a lot of times smart people, right?
People care.
If they go to particular websites more often than once or regularly, they'll put them in bookmarks.
So, you click on the website, right?
You don't actually enter it, right?
How often do we actually enter it?
Yeah, basically, enter it.
So, we're concerned with the authenticity of the website.
There's no human there, usually, right?
It's a website.
There's no human on the other side.

There's no Bob.
Right?
It's a website.
So, we want to make sure we're going to the right website, the authentic website.
That's why the website authenticating to us, to our browser, is important.
How important is this to authenticate us to the service?
Well, yeah, this is like a social network or another bank where you have an account, yeah?
Clearly.
Yes?
It's important.
But does your browser know how to authenticate as you?
No.
No.
You authenticate the human using a password, a PIN, an MFA, or something like that.
A YubiKey.
So, client, the real client to web server authentication happens at a higher level, or higher level.
Userless.
For the client, stay outside, please.
For the client, it doesn't make sense to authenticate to the server, the same way the server authenticate the client.
Moreover, a lot of web servers we use don't have user accounts, or at least don't require.
If I go to WWCIDU, I'm going to require an account.
Does it say you log in?
No.
There's a lot of public information out there.
If I go to WWWhiteHouse.gov, a lot of public info.
But I care that I go to WWWhiteHouse.gov, not WWWhiteHouse.com, because that used to be a porn site.
I don't know what it is now.
So, yeah, I want to make sure I go to the right place.
Right?
And so, UCI.edu, not UCI.org, which is United Cycling International, which exists.
That's the other UCI.
So, what I've said, that justifies the reason why SSL is so leaning on the security of web servers and much less concerned with the web client.
Okay.
See you Tuesday.
I was wondering.
Yeah.
What did you decide?
It's T8, no?
Oh.
It's T8.
So, he asked the professor.
That's the thing.
Yeah.
Last week.
Yeah.
Yeah.
Oh, yeah.
Right there.

Yeah.
Don't worry about it.
You know.
It's not their responsibility to worry about it.
Okay.
They're responsible.
You know.
Okay.
They're responsible for funding.
Because you have an awful lot of the states that are getting this.
Yeah.
But it is .
Oh.
Sorry.
I didn't manage to pay attention to that.
Oh.
Yes, it is.
You know.
It's fine.
Disprint's fine.
I have WeChat.
Oh, WeChat is easy.
Yeah.
I've used it for your projects before.
Yeah.
Okay.
Let me .
Yeah.
Okay.
Hey.
You.
You.
You.
Yeah.
These are the two suspects that you want to work with?
Yeah.
Yeah.
And figure out how you're going to make your project more substantial.
Right?
Now you have a trio rather than a duo.
Okay?
Right.
Okay.
All right.
But.
No.
What I want is an actual update.
So.
Kind of like extend your proposal.
ASAP.
You know.

Take you into account.
Like.
You know what I'm saying?
No?
Yeah.
They send me a proposal.
You are not part of it.
Yeah.
That's what I'm trying to do.
Yeah.
Now that there's you.
There's more.
It has to be done.
Yeah.
Okay.
Right?
You don't get them.
You don't get a free bath.
Yeah.
Yeah.
Yeah.
So basically that's all I want.
真的很 quick.
So please like to let it go.
Do it at number.
Come on.
Do it at number it Er.
Good.
Ummm.
Yeah.
I'd actually bring it over.
The disc...
Why don't you guys.
Why don't you pop them up?
No I don't.
I don't want them on.
They're doing so...
damn it I don't showing them...
...
...
...
...
Thank you.
Thank you.
Thank you.

Thanks for sharing the transcribed lecture — I've **carefully read the SSL/TLS section** of the transcript. Here's a **summary with focused insights** relevant to your research or study:

🔐 SSL/TLS Summary from Lecture Transcript

1. General Background

- SSL = Secure Sockets Layer
- TLS = Transport Layer Security
- TLS 1.0 ≈ SSL 3.0 (essentially the same, renamed due to trademark issues)
- Defined in RFC 2246

2. SSL/TLS Sits Between TCP and Application Protocols

- Most commonly used with HTTP → results in HTTPS
- Built into modern browsers and web servers
- Can be used to secure **any TCP-based application protocol**

⚙️ Protocol Operation

SSL/TLS provides **secure communication** through:

- **Authentication** (typically server-only)
- **Confidentiality** (via encryption)
- **Integrity** (via MACs)

SSL/TLS involves:

- A **Record Layer**: handles packet-level data encryption and authentication
- An **Upper Protocol Layer**: handles handshake, alerts, cipher negotiation

---

🤝 SSL Handshake Highlights

- **Public key cryptography is only used initially** to establish a secure connection
- After key exchange, everything switches to faster **symmetric cryptography**
- No third-party authority like Kerberos (e.g., no KDC); instead relies on **X.509 certificates** and **PKI**

Typical handshake steps:

1. **ClientHello**:
    - Client offers TLS version, random nonce, and supported cipher suites
2. **ServerHello**:
    - Chooses cipher suite, sends its certificate, and server nonce
3. **ClientKeyExchange**:
    - Client sends a pre-master secret encrypted with server's public key

4. **Master Secret Derivation**:
   - Both sides derive session keys from the pre-master secret + nonces
5. **Finished messages**:
   - Each side sends a MAC of all prior handshake messages using the derived keys

---

🔑 What is Computed Using `master_secret`?

1. **Session Keys (Key Block)** for:
   - **MAC** keys
   - **Encryption** keys
   - **IVs** (if using block ciphers like AES-CBC)
2. **MACs for Finished Messages**
   - A critical step ensuring integrity of the handshake itself

   The `master_secret` is computed from the `pre_master_secret` and the two nonces (client + server). From the `master_secret`, both the client and server derive identical session keys to be used during the secure communication.

🛡️ Authentication Focus

- **Server authentication** is standard (via certificate)
- **Client authentication** is rare and optional (e.g., client certs)
- Browser security model centers around **ensuring the user is talking to the right server**
  - Mitigates phishing, spoofing (e.g., `whitehouse.gov` vs `whitehouse.com`)

💬 Prof's Key Takeaways

- TLS/SSL is **stateless** compared to Kerberos (no persistent session storage)
- There is **no trusted third party like KDC** — the server proves identity via certs
- The **client-side authentication** (user identity) usually happens **later at the application level** (e.g., login forms, MFA)

✅ Summary Thoughts

- SSL/TLS leverages **initial asymmetric crypto** to bootstrap **efficient symmetric crypto**
- All security-critical data flows (credit cards, login credentials, etc.) occur **after handshake**
- TLS is **ubiquitous and essential** for modern web security

---

Would you like:

- A **diagram** showing the handshake flow?

- A **comparison table** between SSL/TLS and Kerberos?
- Or help writing a **modeling section** that explains how `master_secret` leads to session keys in LaTeX?

Let me know how you'd like to follow up.

explain the concept of session and connection?

Great question! The concepts of **session** and **connection** in **SSL/TLS** are foundational but often misunderstood. Here's a clear and intuitive explanation:

🔄 SSL/TLS: Session vs Connection

🔹 Session

A **session** in SSL/TLS is a *set of shared cryptographic parameters* between a client and server.

- **Created during the handshake**
- Contains:
  - `master_secret` (derived from pre-master secret)
  - Cipher suite (algorithm choices)
  - Compression method (if any)
  - Session ID
- **Purpose**: Avoid repeating expensive public key operations when reusing secure channels

✅ **Can be reused** for multiple connections (called *session resumption*)

🔹 Connection

A **connection** is a *single, secure communication channel* established between the client and server using parameters from a session.

- Uses:
  - Symmetric encryption keys (from `master_secret`)
  - MAC keys (for message integrity)
  - Sequence numbers (to prevent replay attacks)
- Short-lived: one connection = one encrypted tunnel

✅ A session may spawn **many connections**, but each connection has:

- Its own sequence numbers
- Its own key states (MAC/IVs/etc.)

📦 Analogy: Library Membership vs Visit

Concept	Analogy	SSL/TLS Meaning
Session	Library membership card	Cryptographic agreement from handshake
Connection	A visit to the library	A single secured exchange (e.g., one HTTPS session)

🔁 Session Resumption

Instead of repeating the full handshake:

- The client sends a **Session ID** from a previous session.
- If the server still remembers it, they **resume** using the same `master_secret` (or derive new keys from it).
- Faster, less overhead.

💡 In summary:

Term	Scope	Established By	Contains	Lifespan
Session	Persistent context	Full handshake	master_secret, cipher suite	Long (cached)
Connection	One data tunnel	Derived from session	Symmetric keys, sequence numbers	Short (per exchange)

Would you like a **visual diagram** showing how sessions and connections relate across multiple HTTPS requests?

In the handshake protocol, so how does client validate the public key of the serve server? The slide mentions that there is something called a server SART key. So can you explain how this process happens?

Excellent — you're getting into the **heart of SSL/TLS server authentication**, which hinges on **public key validation** via **X.509 certificates** and a **chain of trust**.

Let's walk through **how the client validates the server's public key** during the TLS handshake, and what the **"server cert key"** really refers to.

🔐 Step-by-Step: How the Client Validates the Server's Public Key

🔹 1. Server Sends Its Certificate

- In the `ServerHello` message, the **server sends a digital certificate**.
- This cert contains:

    - The **server's public key**
    - The **server's domain name** (e.g., `www.uci.edu`)
    - A **digital signature** by a trusted **Certificate Authority (CA)**

✅ This is what the slide refers to as the **"server cert key"** — it's the **public key embedded in the certificate**.

🔹 2. Client Validates the Certificate

The client browser or application performs several checks:

✅ a. Signature Check

- The client:
    - Extracts the **CA's name** from the certificate
    - Looks up the CA's **public key** in its **trusted root store**
    - **Verifies the CA's signature** on the server's cert

If the signature is valid, this proves:

> "A trusted CA vouches for this server's identity and its public key."

✅ b. Hostname Check

- The domain name in the certificate (e.g., `CN=www.uci.edu`) **must match** the domain the client is connecting to.
- This prevents **spoofing or phishing attacks**.

✅ c. Validity Period Check

- Is the cert expired or not yet valid?

✅ d. Certificate Revocation Check (optional)

- Uses OCSP or CRLs to check if the certificate was revoked by the CA.

🔹 3. Client Trusts the Public Key

If **all checks pass**, the client now **trusts the server's public key**.
This key is then used for:

- Encrypting the **pre-master secret** (in RSA-based TLS)
- Or verifying the server's **ephemeral key signature** (in ECDHE/DHE)

🧠 Recap: Why This Works

SSL/TLS doesn't trust the server blindly — instead:

- The server provides a **certificate**
- A trusted CA has signed it
- The client **verifies** the chain of trust, domain name, and time validity
- Only then does it use the **"server cert key"** for key exchange

---

🔒 Real-world example:

Your browser stores trusted root CAs like:

- DigiCert
- Let's Encrypt
- Google Trust Services

If the server's cert is signed by one of these (directly or through an intermediate), the browser accepts it. Otherwise — you get a **"certificate warning"**.

Would you like a **diagram** showing this certificate chain verification and how the client uses the server public key during handshake?

> I think the professor gave an example using the UCI's website and how a client can validate the server certificate sent from the uci.edu domain. Can you give that example?

Yes — your professor gave a great **real-world example using `www.uci.edu`** to illustrate how a browser (the client) validates the server's certificate using **certificate chains and domain checks**. Let me walk you through that **UCI example** in the exact spirit of what was said:

🧾 🔐 Example: Connecting to `https://www.uci.edu`

✅ 1. Client Initiates Connection

- You open your browser and enter `https://www.uci.edu`.

✅ 2. Server Responds with a Certificate

- The UCI web server sends its **X.509 certificate** as part of the TLS handshake.
- That certificate says something like:

```vbnet
Subject: CN=www.uci.edu
Issuer: DigiCert TLS RSA SHA256 2020 CA1
Public Key: [UCI's actual RSA or ECDSA public key]
Signature: [signed by DigiCert]
```

✅ 3. Client Verifies the Certificate

The browser now performs **three main checks**:

🔍 a. Signature Chain Verification

- The certificate is signed by DigiCert CA (a known and trusted Certificate Authority).
- The browser checks: "Is DigiCert in my list of trusted root CAs?"
  - Yes → Signature is valid
  - No → Show a certificate error

🔐 The certificate is signed using the CA's private key; the browser verifies it using the CA's public key.

🔍 b. Domain Name Matching

- The certificate says: **`CN=www.uci.edu`**
- Your browser is visiting **`https://www.uci.edu`**
- ✅ So the **domain matches** the certificate

If it had said **`CN=www.random.com`**, your browser would raise an error (e.g., **"certificate does not match the site"**).

🔍 c. Expiration Date Check

- The cert is valid from, say, Jan 2024 to Jan 2025.
- Today is within that window → ✅ OK
- Otherwise → ❌ "certificate expired or not yet valid"

✅ 4. Conclusion: Public Key is Trusted

Once all these pass, the browser **accepts the certificate** and **trusts the public key inside** it.

Now it can:

- Encrypt a **pre-master secret** to UCI's public key (RSA), or
- Verify a **digital signature** if ephemeral keys are used (ECDHE)

This lets both sides **derive symmetric session keys** for secure communication.

🧠 Professor's Key Point

> You don't just blindly trust the server — your browser ensures that:
>
> 1. The cert is signed by someone **you already trust**
> 2. The cert actually belongs to www.uci.edu
> 3. The cert is **currently valid**

Only **then** is UCI's public key accepted and used for encryption.

Would you like to see the actual certificate for `www.uci.edu` using OpenSSL or your browser? I can walk you through that too.

what is this MAC computed with master secret so what is this MAC actually

Great question — you're referring to the **MAC computed with the master secret** in the TLS handshake, particularly in the **Finished messages**. Let's break down what this MAC is, where it's used, what it's computed over, and why it matters.

✅ What is the "MAC computed with the master secret"?

🔐 It refers to the Finished message authentication:

At the end of the TLS handshake, both the **client and server send a `Finished` message**, which includes:

> A **Message Authentication Code (MAC)** computed over **all prior handshake messages**.

This is the **first message encrypted and authenticated** using keys derived from the `master_secret`.

📦 What exactly is included in this MAC?

➤ The Finished message contains:

```ini
verify_data = PRF(master_secret, finished_label, Hash(handshake_messages))
```

Where:

- `PRF` = Pseudo-Random Function (based on HMAC)
- `master_secret` = shared secret from the handshake
- `finished_label` = "client finished" or "server finished"

- `handshake_messages` = **all messages exchanged so far** in the handshake, in exact order and byte form
- `verify_data` = final MAC-like hash sent to the peer

✅ This value is sent **inside the encrypted TLS record**.

🧠 Why is this MAC important?

🔐 It proves:

- Both sides **derived the same master_secret**
- The **handshake messages were not tampered with**
- The party on the other side is **not an impostor** (e.g., a man-in-the-middle)

This MAC serves as a **final integrity and authenticity check** before moving on to secure data transfer.

🔄 Analogy

Think of it like:

> "We just exchanged a bunch of messages. Here's a hash of everything we said so far, using our agreed-upon secret. If yours matches mine, we both know we're on the same page — and no one's messing with us."

🔧 What is the MAC *not*?

- It's **not** the record layer MAC used for data packets (though that also uses master-secret-derived keys)
- It's **not** user data; it's purely for handshake integrity

---

✅ Summary: TLS Finished MAC

Item	Meaning
Computed over	All prior handshake messages
Key used	Derived from `master_secret`
Purpose	Verify handshake integrity & agreement
Algorithm	PRF (HMAC with SHA-256 or stronger)
Sent in	Encrypted `Finished` message
Verified by	Other side (client verifies server's MAC; server verifies client's)

Would you like to see the **equation in LaTeX**, or a diagram showing how this MAC fits into the handshake timeline?

what did the prof tell about ssh in lecture? include everything

The professor mentioned **SSH (Secure Shell) only briefly** in this lecture, but here's **everything he said** about SSH, accurately extracted and summarized:

🔐 SSH Mentions in Lecture (Verbatim + Context)

1. **Name Expansion**:

> "SSH is Secure Shell."

🔹 This was mentioned alongside:

- SSL: Secure Sockets Layer
- TLS: Transport Layer Security

Context: The professor was introducing common secure protocols used on the internet today. He asked if students knew what these acronyms stood for and clarified each one.

🔍 Placement in Lecture

- The SSH mention came **immediately after the discussion on SSL/TLS acronyms**.
- It was **not explained in technical detail**.
- There was **no follow-up discussion or protocol breakdown** for SSH in this lecture.

❗ What was *not* covered:

- No description of how SSH works
- No mention of public/private keys in SSH
- No comparison between SSH and SSL/TLS
- No discussion of SSH authentication (e.g., password vs key-based)
- No explanation of SSH use cases (e.g., remote shell, file transfer)

✅ In Summary:

The professor only said:

> "SSH is Secure Shell."

...as part of a **quick acronym overview**, without elaborating on its structure or use.

Would you like a proper comparison between **SSH vs SSL/TLS** for your notes? Or an explanation of how **SSH key-based login** works?

this is the next lecture, read it oo

I'm Drew Zee, if you don't die, no excuses.
Now, let's share a quick show of hands.
Do you want me to post a few example problems?
Yeah.
From prior years?
Yes.
Let me ask this.
How many don't want them?
I'll throw it together, a few examples.
But I won't post solutions.
Because many of the type of exams I give in classes I teach
generally have more than one solution.
Rarely there's just one.
It's like for undergrad when I teach crypto,
there's just one solution.
It's a numeric answer.
But other than that, the type of questions you have,
that will require one, maximum two paragraph answers.
Okay?
Do not write me an essay.
Okay?
Please.
Save yourself, save me.
Don't write an essay.
Okay?
Also, just as a rule,
there are no devices allowed in the essay.
The phones, calculators,
everything that you normally bring
has to be inside your backpack
or somewhere else away from you.
Okay?
No virtual reality AR type devices like this one.
Okay?
No earphones.
No headphones.
Right?
None of that.
No smart devices whatsoever on your body
at the time of your exam.
Nice.
So your backpack, whatever stuff,
inside, next to you, under you, whatever.
No problem.
Okay?

You need to bring something to write with.
Pen?
Pencil?
Both?
Sure.
Okay?
I will give you paper.
I know.
It's old-fashioned.
It's 20th century.
You write it on paper.
Okay?
I will give you an exam sheet.
You will have some number of problems.
Not too large.
Not too small.
Same rules apply to the final type.
And you will have blank sheets of paper
that I will bring.
And I will give you a couple to start with.
If you need more, you'll come up.
Make some up.
You can write on one side or both sides.
I don't care.
The only thing I absolutely care
is that you do your best penmanship.
I am not here to decipher encrypted text.
It's not what I paid for.
That's beyond my call of duty.
So, if I can't read it,
I'll assume it's not decipherable
and it requires a deduction of points.
I mean, I don't want to be calling everybody in
and saying,
what did you mean here?
What is this encrypted word?
What is it in scrambled sentence?
Please, write legibly.
If you have to print,
use printed letters.
I don't care.
Just write legend.
Okay?
And if you erase,
make sure you erase well
because sometimes, you know,
when you erase something,
it's smudgy
and then you write on top of it
and then it's hard to read.
I'll give you more paper.

No problem.
Okay?
So, that's the exam
and it will cover everything we covered
including today.
I might ask you questions
about the background stuff
from that,
remember that lecture one background?
Why, like, not specific?
Like, not in big detail, right?
It's not relevant to what we covered.
Otherwise, so far,
I won't ask you, like,
how does elliptic curve DSA work?
So, I'll be reasonable
and you'll try to be reasonable.
Yeah?
Are we going to expect
to know, like,
the specific numbers
associated with different protocols?
What do you think?
I don't know.
I'm not happy.
No.
No.
If I ask you something like,
okay, so how does,
let's say today's lecture
was more about IPsec.
So, I'll say,
how does IPsec parse headers?
You should know how it does it
but you don't need to know
the vertical numbers
and, you know,
this kind of thing.
That's minutia.
I don't care about that.
I do not hear.
You're not here to, like,
for growth learning
to memorize some boring numbers,
you know.
Okay, so, on the last episode
of 203,
we went for SSL-TLS
and I know you're probably
more confused than you were before
but you do know a bit more about it

than you knew before.
It is a complicated protocol.
Nobody learns SSL-TLS
in one lecture.
Okay?
It's an effort that took years
by a whole lot of people,
smart people, right, mostly.
people with degrees
and experience
and security experts
to put together this protocol
and to evolve.
So, don't expect to, like,
understand everything
in one lecture
or even two.
What I give you here
in almost every lecture
is essentially
an appetizer
and if you're interested
in the meal,
you go further
and do a dig on your own.
Okay?
So, you will know
just enough to be dangerous.
but it's not enough
to be any kind of an expert.
So, remember,
what the key things
about SSL
and SSL-TLS
is that they operate
at this thin sublayer
above transport.
They are primarily oriented
towards the web
or HTTP traffic
but not exclusively.
People use SSL-TLS
for other things.
But the biggest consumer
is the web traffic.
So, we covered
the handshake, right?
Remember
the handshake protocol?
Right?
That's the main

thing about CLS
is without the handshake
nothing else works.
The handshake protocol
is very important.
It is
a full handshake
meaning that
it requires
an initiator,
the client,
to send something first
usually not authenticated
because the first message
cannot be authenticated
if there's no prior
contact.
And then the server replies
also with usually
not authenticated message
saying,
here's my certificate
and then the server
already computes
a session ID
because remember
the client supplies
a random number
a nonce
in the first message
client hello
the server replies
with a certificate
and the server hello
and that contains
a nonce.
So, given the two nonces
and some other stuff
they have a session ID
so they can identify
this particular session
between this client
and this server.
Okay?
And recall that
99%
more than 99%
of the cases
client authentication
isn't done
in SSL TLS.

It can be done
but it isn't done
for reasons
that I already discussed.
Right?
And
most of the time
today
you will see
that most
key exchange
right?
Not
right?
Key exchange
is done
using RSA
where the client
picks what's called
pre-master secret
encrypts it
using the server's
public key
all this is
after you check
the server certificate
right?
You don't do this
right?
The client abandons
the protocol
if there's something
wrong with the
server certificate.
Do you remember that?
Yes?
So the idea
in security protocols
is you don't
do heavy work
before you do
light work.
So the light work
means you make sure
that the server name
in the certificate
matches the URL
the domain name
of the URL.
Otherwise
you're being phished

and attacked.
Okay?
You're being scammed.
So test the map.
Of course
it doesn't help you
if you mistyped the name.
If you went
wanted to go to
wellsbargo.com
and you typed
wellsbargo.com
well guess what?
If somebody
if somebody
has a website
called wellsbargo.com
and registered it
and has a certificate
that's where you're
going to go
and SSL TLS
will not help you.
It doesn't sound
too good
does it?
That's called
that kind of attack
was called
type of squatting
where
adversaries
or sometimes
not even adversaries
are like
porn sites
or like
marketing sites
will set up
a domain name
a domain name
that closely
resembles a
legitimate
well-known
domain name
hoping that
the users
will mistype
the name
and they will

land on their
site.
In the worst case
you will see
some questionable
content
or you know
a walking tub
will be advertised
to you
by Akron.
In the worst case
you'll be fished.
Okay?
Meaning that
they will present
to you something
that looks like
a wellsbargo page
and you'll
blindly
most people
blindly
type their
username
and password
and that's it.
Bye-bye.
And so
SSLTLS
won't help you
with this
but if you
click
let's say
on a bookmark
that you put
a long time ago
that says
wellsbargo.com
SSLTLS
will help you
because there's
no typos
right?
Right.
So most
common
in
TLS
before

version 1.3
remember
I told you
there's like
two real
versions
out there
today
there's
really
TLS 1.2
and 1.3
they probably
used
the most
there are
lower versions
older versions
still in use
but they're
being deprecated
so a lot
of web
servers
will simply
not accept
a conversation
and TLS
session
with a client
that does
not support
at least
1.2
TLS
now 1.2
and 1.3
differ
I already
alluded to
that
but my
discussion
last time
was focused
on 1.2
so in
1.3
unlike here
you can
no longer

use RSA
to encrypt
okay
in 1.3
you have to
basically use
signatures
and use
Diffie-Hellman
or something
similar to
Diffie-Hellman
for key exchange
you cannot use
RSA
for key exchange
right
we looked at
how key is
derived
right
because remember
there's
pre-master secret
then there's
like a
master secret
well
in TLS 1.3
I believe
there's no
pre-master secret
because actually
both sides
contribute to
the key
and compute
what's called
the master secret
because Diffie-Hellman
is symmetric
symmetric
meaning both sides
contribute equally
and then
using the master
secret
we derive
using
something called
the k-derivation

function
a set of
other keys
right
that we actually
use for
encryption
and authentication
during the
data transfer
okay
we looked at
this
this
this
and then
there was
a summary
right
we stopped
around here
no
this is a
summary
of
a
handshake
client
below
followed by
three messages
essentially put
together
server hello
server
certificate
change
server hello
down
then
client returns
to server
saying
okay
this is client
key exchange
message
right
so now
that message
can be secured

already
client finished
by the way
contains
in mac
of all
of everything
exchanged so far
computed
with a key
that they just
agree on
master secret
key
and the last
message in the
exchange
is server
goes to client
says confirming
that we selected
this
I mean this
wording is confusing
change
they're not changing
any cyphers
today
right
they already agreed
on what cyphers
they use
important one
is server
finished
that can also
contain
a mac
of all the
messages exchanged
from the
service
perspective
so if the
survey of the
client have
the same view
of messages
sent and received
that means
no one

has interfered
with the
protocol
not a single
bit has
changed
then they're
sure that
they're talking
to each other
and that
completes the
handshake
we talked
about this
talking about
this
there's some
options
yes
so the other
thing you can
do
is if
client and
server have
previously
established
SSL-TLS
connection
there is already
a session ID
remember there
was a session ID
derived
that identified
the SSL-TLS
connection
the client
can resume
the connection
yeah
resume
means
hey
I'm still
here
I'm still
here
there's two
purposes

for that
this one's
kind of like
saying I'm
around
don't
don't
don't
flush
the state
right
I'm still
here
and the
other one
is
the other
purpose
is
oh
let's
change
keys
let's
refresh
key
okay
so
if you
have
what's called
session resumption
right
you can
essentially
maintain the
same
not have to
redo the
handshake
right
you can
maintain the
same
master secret
but the
nonsense can
change
remember
nonsense
these

randoms
they can
change
and so
the actual
encryption keys
and authentication
keys will change
as a result
you can
theoretically
also change
the cipher suite
at that point
most of the time
that doesn't happen
but you can
change
say ah
we'll be using
I don't know
RSA
with
AES
let's switch
to something
else
you could do
that
if the
policy allows
if both
sides
agree
you can
do
never mind
this
so
the main
I'm not going
to go in
such detail
over 1.3
it is actually
in some ways
a simpler
protocol
let's
zoom
on this

this is a
handshake
so what we've
seen is this
right
the first
four messages
are the handshake
and then you get
the HTTP
get
HTTP
right
because this
is what you
really want
right
you want
to fetch
a URL
so this
and this
gives you
like
this timeline
gives you
approximate
like
for comparison
an average
timeline
for how long
it takes
so if you look
at
it starts
with 0
and
it's 200
milliseconds
until you get
here
okay
that's two
round trips
and that's
pure overhead
and so now
it's pure overhead
as far as
the user

is concerned
and then the
real useful
stuff is
get and put
right
here
now with
version 1.3
there's just
one round trip
and then
two packets
plan
hello
key share
and then
server
hello
key share
certificate
certificate
verify
blah blah blah
blah blah
okay
we're not going
to go
through the exact
format
but
essentially
the time
is cut in
half
with pretty
much
the same
result
that the
two parties
there's still
a handshake
it's just
a two message
handshake
not a four
message
handshake
and it's
faster

obviously
right
so 100
milliseconds
you might
not really
care
but over
time
this adds up
right
for many
users
over time
right
a lot of
traffic
a lot of
delays
so cutting
things in
half is
better
you had a
question
what's the
improvement
that
okay
many
there are
a number
of improvements
one of the
things is
simplification
so
TLS
1.2
and B4
they have
way too
many
cipher suites
some of
them
got deprecated
because they
were like
RC4
is a

cipher
MD5
a hash
function
they are
broken
or at least
they are
no longer
secure
DDS
triple
DES
no longer
considered
secure
or too
expensive
I forget
exactly
yeah
I think
it's
no longer
considered
secure
so
most of
the
time
today
you
will
see
AES
as
the
encryption
as
the
bulk
encryption
you
will
see
SHA-256
you
will
not
see
SHA-128

anymore
so
rather than
hundreds
of
cipher suites
I think
the
main ones
are like
five or six
today
and
they're not
using RSA
anymore
so
also
the names
of the
cipher suites
changed
like
I remember
I was showing
like
TLS
RSA
with
blah blah blah
blah blah
and
the first
thing
that
this
string
would
tell you
is
how
are we
exchanging
keys
which is
how
RSA
when you
client
increase
a random

number
for the
server
and the
second
one
was
how
does
server
authenticate
also
RSA
because
he
decrypts
right
and
something
that the
client
sent
to him
and
can
implicitly
authenticate
so
that's
gone
instead
of
that
they
no
longer
say
how
you
exchange
the
keys
everything
is
done
essentially
by
a
DPA
element
or

some
version
of
DPA
element
which
is
either
elliptic
curve
or
normal
and
the
signatures
are now
required
so
there's
no
more
implicit
authentication
so
now
both
you
know
both
clients
and
the
server
have
to
stop
well
the
server
has
to
stop
basically
and
the
client
can
sign
with
the
ephemeral

key
they
just
generated
okay
so
it's
a
strictly
stronger
syntax
strictly
more
higher
security
at
half
the
cost
so
that
higher
security
enables
the
reduction
in
messages
no
it's
just
no
not
not
not
higher
no
it's
a
reduction
in
messages
and
shading
off
some
of
the
questionable
currently
questionable

algorithms
allows us
to do
this
because
what the
client
can do
now
you see
the client
hello
is still
in the
form
the client
says
I don't know
anything about
this server
right
I just
know the
URL
so what I want
is I want a
certificate
that says
that this
URL is inside
the certificate
and it's signed
by somebody
I trust
right
so
it's like
essentially
you come
to
I'm a
I'm a
hiring manager
at a company
you come to me
and you say
I have a UCI
degree
I may not have
ever heard of UCI
okay

and I've never met
you
but if you give me
a signed
diploma
signed by UCI
register
and then I look up
UCI register
and I see that
they are certified
by University of
California
Central Authority
and that one
is certified
by a California
Department of Education
well
I trust
the Department of Education
of the state
of California
you see what I mean
so I know
their root certificate
I will verify
the UC certificate
then I will verify
UCI certificate
then I will verify
your certificate
make sure it's valid
make sure it's not revoked
right
that kind of thing
so that's still the case
the client
before this starts
does not trust
the service
just knows the URL
and so
that remains
in both versions
the web
the fundamentals
of the web
don't change
but what we're saying
now is a client

for example
can generate
like
think of it
as a temporary
public key
there
right
where he knows
the private key
and what he does
is he can now
send a message
with a public key
to the server
and he can say
and by the way
I do know
the private key
and how can you do this
he can sign
with a private key
that message
that includes
the public key
kind of it was
essentially like
a self-signed certificate
the server said
oh look that's cool
because now
you're not joking
you're not messing around
with me
you know the key
which you
you know the private key
for which the public key
you'll be just sending me
so then the server says
okay now I have
my own certificate
here's my certificate
oh maybe a chain
of certificates
and I only use RSA
certificates can be signed
with RSA
but my public key
is no longer RSA
it's some kind of

D.K. Hellman type
public key
which can be used
for many things
but
there it is
so now
a server has a public key
from the form
remember G to the B
or something
and the client
has a public key
remember G to the A
and then they can
each compute
G to the A B
okay
so that's basically
the idea
and
when the server
receives that message
one
he's able to
compute the key
right away
because
he knows his
you know
he knows B
right
that's his private key
and he receives
G to the A
so right away
he knows it
and in that second message
the second green message
he can already sign
using that key
or MAC
okay
so that message
can be secure
they already have
a secret
in common
so
what is
yeah

I would like to know
for the client
when the client
decide to
choose the
TS 1.2
and 1.2
so
so
this is
the client
hello
did you get
that announcement
I posted
yes
did you see
that announcement
I posted
about this
there's a very cute
little site
that if you expand
everything
it will tell you
in excruciating
detail
everything
you want to know
about this
now
if you
in that pointer
that I sent you
link
right
there's a 1.2
if you substitute
with 1.3
you will see
the same
gory details
of a 1.3
handshake
okay
now
the difference is
the client
hello
is present
in all of them

so
at the client
hello stage
the first thing
that the server
sees is
the setup
or the highest
version
that the client
is supporting
so if the client
specifies
the highest version
he supports
is 1.2
that server
says
okay
if I'm allowed
for example
the server
could be
federal government
saying
go away
I don't
I don't talk
to anybody
who doesn't
speak the latest
version
okay
but if it's
an e-commerce
site
they're like
I don't
sell this one
I don't care
if you're using
the latest version
so it will say
okay
1.2 is fine
you see what I mean
if the client
says
oh
1.3
in the client

the server
could say
yeah
but I'm in
I don't know
Slabonia
and we don't
speak 1.3
here yet
we're still
like living
10 years ago
so then
we regress
to 1.2
you see what I mean
so server will
reply with
1.2
but yeah
for the previous
slide
yeah
for the previous
slide
it seems that
the client
sent more
message
if it
uses
1.3
so if
no
no
less
less
because
look
both of them
say
client
hello
right
but the client
hello
is not exactly
the same
if you look
at the
that's why

I told you
if you look
go into that
website
right
and you compare
the client
hello
messages
for both
1.2
and 1.3
they're not
exactly the same
but the beginning
is the same
do you see what I mean
they are the same
in the beginning
so
if you
if the client
hello
is a 1.2
hello
the server
will realize
that
you see what I mean
so if the client
is willing to speak
1.3
the client
hello
should be 1.3
if the client
hello
the client speaks
only 1.2
up to 1.2
he will
it will be 1.2
but the server
will recognize
by looking
in the first byte
yeah
so
no
because
the server

will simply
reply that
my check
cipher spec
I will pick
1.2
now
don't quote me
on this
but I think
one is a superset
of the other
so it's still
parsable
you don't have
to trust me
you have to trust
all these hundreds
of people
that work
on this
to make sure
because it's all
made for programmers
right
it was all
made for people
to make it easy
to code
to parse things
right
so even if
the format
changes
the preamble
almost never
changes
right
so you should
be looking
at like
the very
first few
fields
in the header
and say
ah
that will tell
me the version
right
and if I don't

support this version
then I can't
then it's an error
but if I can't
support this version
then we can
downgrade
right
if my policy
allows me
right
so for example
maybe
that depends
very much
on web
web
web
web
web service
policy
whether it's
like a client
might come
from
within the
US
and say
I speak
1.2
and the web
server says
okay
I'll speak
1.2
with you
and the client
comes over
from Botswana
and the server
says
uh-uh
I don't
trust Botswana
I don't know
where the hell
you are
really
so for foreign
clients
I will

bunk with
three or nothing
it's totally
fine
to do that
so primarily
what do we
use this
to sell
for
I mean
the primary
thing
has always
been
some kind
of e-commerce
right
because
we're buying
stuff
right
internet
was built
essentially
on e-commerce
internet
of today
not
the original
internet
I would say
sorry
correction
the web
right
the driving force
of the web
is e-commerce
so
as I
already motivated
before
client-less education
is generally
not needed
why
well because
with what
right
you're going

to an e-commerce
site
you're buying
widgets
from
ADC
incorporated
and you may
have never
visited
ADC
incorporated
you're
bargain shopping
right
using google
saying compare
prices
oh look at
that
a company
in Kravistan
all right
cheap stuff
let me get
it
they don't
know you
you don't
know them
okay
so you
authenticated
them
but why
should you
authenticate
yourself
as a client
until you
decide to
buy
and it's
not really
authentication
that you're
doing
right
you're just
presenting
a method

of payment
right
which may be
stolen
as long as
it's valid
the web server
doesn't care
all they want
to make sure
is that you
will pay
who you are
they don't really
care
so if you're
sending credit
card information
over
of course
that should be
protected
so that's what
really SSL is for
now
it's also there
to protect
to you
against evil
web servers
and that's
where it
transcends
e-commerce
when you go
to CNN.com
assuming you
trust CNN
or whatever
your taste
may be
fox.com
you want
to get news
from them
and not
from some
belorussian
government agency
right
or a north

korean
mouthpiece
of what's
his name
you know
so you
want to get
the real
thing
so in that
case it's
not money
that's a
stake
it's authenticity
right
in that case
nobody cares
that you
actually encrypt
right
in that case
you may not
even care about
encrypting the
data
you care about
authenticating
the server
maybe using
null encryption
make sense
right
of course
SSL
and TLS
they're not
magic
right
they're not
going to do
anything
about like
guaranteeing
that if you
send credit
card data
and shipping
address
or whatever
they're not

going to sell
it to some
mass marketer
or some
annoying
telemarketer
or just
give it to
somebody on
the street
to quickly
buy 500
pair of jeans
you know
you have no
idea what's
happening
with your
credit card
you're just
going to
trust them
that they
will erase
it
or store
it
securely
most
clients
of course
have no
idea
what it
means
for a
server
certificate
to even
exist
right
most
think about
not
yourselves
because
you're all
super
educated
folks
but

think
about
various
family
members
and
you know
some of
them
are
children
they don't
know
they use
the web
they use
the internet
they have
no idea
what it
all means
so we
rely on
conscious
correct
implementations
to make
sure
that
certificates
are valid
the certificates
correspond to
the websites
that they're
assigned to
etc
more importantly
than
e-commerce
although that's
not as much
of a segment
for
application
domain
for
CLS
is
banking
and other

financial
services
right
it's kind
of
you can
think of
it as
hyper
e-commerce
right
this is
where the
money
the real
money
is
your
money
right
here the
bank is
not selling
you anything
right
you already
decided to
put your
money in
the bank
and you
have a
relationship
so instead
of entering
your credit
card info
actually here
you do
enter
to begin
a session
with the
bank
you do
enter
your
username
password
and god
knows what

else
MFA
right
so
there's
more
of a
beginning
of a
session
and end
but notice
that people
go to
banks
also to
browse
let's
say
interest
rates
without
having to
log in
so you
do want
to make
sure you're
going to
the bank
it's the
right bank
and it
displays you
truthful
information
so SSL
TLS is
useful there
too
right
even without
you logging
in
and authenticating
yourself
sometimes
in early
2000s
mid 2000s
some banks

that were
kind of
hip
and
cool
thought
they
would
send
customers
I think
I mentioned
before
like various
dongles
and USB
keys
and other
things
that you
could use
to upload
a client
certificate
for SSL
TLS
that way
you could
have
strong
two-way
authentication
this
somehow
fell out
of
fashion
okay
that's
the end
of
SSL
TLS
now we
go on
to
another
beautiful
masterpiece
IPsec
quick show

of hands
how many
people here
have heard
IPsec
two
three
and
well
okay
so
IPsec
is IP
security
extension
strictly
speaking
unlike
Carverus
which is
a system
right
Carverus
remember
e-system
have
many
protocols
right
components
SSL
TLS
is a
protocol
right
our
protocol suite
really
and a set
of formats
right
the protocols
always have
formats
for messages
so
IPsec
strictly speaking
is not
a protocol
it is

actually
a set
of formats
okay
so
this will
become
clear
maybe a
little bit
puzzling
at the
moment
become
clear
in a
bit
I hope
so
let's
consider
IP
IP
is the
lingua franca
the English
of the internet
right
you want
to
communicate
you kind of
better speak
English
in the business
world today
well
if you want
to communicate
today
you need
to speak
IP
you don't
speak
IP
you are
basically
all right
IP
is the

network
layer
it is
what's
called
the
thin
waste
because
the internet
has this
kind of
a
what do you
call it
the hourglass
the hourglass
shape
if you look
at the
protocols
right
there are
many
protocols
why do I
say hourglass
right
so
so
this
will
be
physical
layer
data
link
and
here
is
network
layer
here
is
where
IP
goes
so
here
things
get

thicker
and above
IP
let's
say
this
is
IP
and
above
IP
you
have
transport
layer
session
layer
so
I have a
presentation
application
whatever
I forget
that
but
they're
all
poorly
demarcated
here
you have
lots of
applications
right
lots and
lots of
applications
below that
also
lots of
applications
like
presentation
platforms
of various
sorts
and here
below
in the
physical
layer

you have
Wi-Fi
Zigbee
Bluetooth
NFC
cellular
right
you have
right
all these
fiber
optics
twisted
copper
god
knows
what
oh
radar
infrared
right
all these
ways of
communication
so
there are
many choices
here
many choices
here
as the
network
layer
on the
internet
no
choice
IP
or
nothing
that's
what it's
called
the
thin
waste
so
as I
said
before
IP

was
designed
started
around
1976
the
design
was
finalized
around
1980
or
81
most
of the
work
in
designing
IP
was
done
actually
not
far
from
here
the
place
I
once
worked
it's called
ISI
information
sciences
institute
part of
USC
school
of
engineering
beautifully
situated
far away
from
USC
in
Marina
Del Rey
on
water

inspiring
people
to work
hard
in
course
so
true story
I wasn't
there
I'm not
that old
but
it was
done
a lot
of it
was done
there
at UCLA
also
there were
some
internet
pioneers
who designed
some of
this
and
DARPA
which is a
defense
advanced
research
project
agency
funded
all of
this
and
without
DARPA
there
would
have
been
no
internet
anyway
so
it was

designed
to connect
a bunch
of
routers
switches
computers
and
all
people
well
to
network
and
primary
motivation
back
then
for
the
internet
was
to
access
scarce
resources
remember
back
then
you had
computers
most
of
them
there
were
no
personal
computers
first
of
all
zero
when
FB
was
designed
none
not
such
a

thing
did
not
exist
they
were
mid-range
computers
and they
were
like
mainframes
bigger
computers
and then
the few
here and
there
around
the
world
supercomputers
okay
a
supercomputer
from
back
back
then
is
an
IoT
device
today
in
terms
of
its
computing
power
but
they
were
supercomputers
and
it
took
like
a
whole
floor

OpenAI

of
building
one
computer
because
these
resources
were
scarce
and
super
duper
expensive
like
gazillions
of
dollars
they
were
precious
right
and people
who wanted
to perform
large scale
computation
of any
kind
have to
like
buy
time
on those
computers
and
if
you
do
it
you
travel
somewhere
to
use
your
computer
right
there
physically
but
that's

not
scalable
so
people
realize
that
if
you
want
to
have
access
remote
access
to
resources
like
that
you
need
to
connect
them
somehow
and
dial
up
what
existed
modems
remember
modems
maybe
you've
seen
old
movies
go
and
there's
a
modem
connection
yeah
it's
not
somebody
having
a
cold
well

I
still
remember
that
and
but
that
was
totally
terrible
because
you
were
using
unstable
telephone
lines
and
terrible
equipment
so
there was
the idea
to connect
them all
with
dedicated
lines
okay
so
there you
have the birth
of the internet
right
and they
needed a
protocol
that would
basically
break
communication
data
arbitrary
size
into
small
uniform
size
chunks
and
transmit

them
hop
by hop
by hop
by hop
to their
destination
why
hop
because
you
couldn't
have
fully
connected
network
right
everybody
connected
to
everything
that would
be incredibly
expensive
logistically
nightmare
right
so
they
needed
to
connect
them
via
switches
routers
right
that scales
so
the internet
protocol
was designed
for that
to hop
along
these
routers
and switches
until
these
get to

the
destination
IP
protocol
does
not
actually
perform
the routing
function
it doesn't
distribute
routes
okay
that's done
by what's
called
routing
protocols
which you
should have
learned in
your networking
course
IP just
forwards
things
all it
knows
is how
to
let the
source
generate
the packet
stick it
on the
wire
give it
to the
data link
layer
below
and then
at the
next hop
whatever that
hop might be
it might be
the destination
or it might

be a router
pick that packet
up saying
hmm
let's see
destination
I don't
know this
destination
I am not
that destination
but I think
somebody else
I know
knows where
the destination
is
so you forward
it to somebody
else who knows
better
according to
some routing
data
so back then
the internet
was small
and very friendly
it was populated
by idealistic
and naive
geeks
who just wanted
to compute
stuff
right
and they were
not you know
hostile
I mean they were
they could be
hostile
but they were
not about
to attack
the internet
the cow
that feeds
them all
right
it was all

essentially
research
and almost
no commercial
use was allowed
on the internet
until early 90s
so for good
like 12
14 years
there was no
such thing
as commerce
on the internet
and so
from that
small and
friendly world
we wound up
what we have
today
essentially
internet
in many ways
is a giant
global sewer
where everything
goes
and anything
flies
and back then
all the hosts
were known
in fact
if you were
running a
unix machine
back then
you didn't even
need domain name
service
you could have
what's a file
called
etc password
there was a file
called etc hosts
and it actually
listed one per line
names
and IP addresses

of every host
on the internet
yeah
I remember
when I inherited
my first
machine
in grad school
it was like
from
the machine
was older
but it had
this file
and it had
like I don't know
5,000 entries
and there were
all the hosts
on the internet
that's it
everywhere in the world
the users
were limited too
right
security was
essentially
not an issue
back then
in that world
it was not an issue
think of it
as the internet
was essentially
like a hippie
hippie-dippie
private network
you couldn't
just like
randomly hook up
to the internet
there was no
wireless
access
zero
okay
the easiest way
to work with
the internet
was to actually
like

use a
wired
ethernet
right
you have an
ethernet port
in the wall
you
start with
wire
and
if you have
internet access
you can start
doing
well
as you know
the world has
changed dramatically
since then
so
security has
become an issue
now
people were not
totally naive
already
in the late 80s
they started
thinking about
what will happen
when the internet
becomes available
to the masses
and becomes
commercialized
so
hostility
or
essentially
lack of security
in the original
internet
was already
a problem
so what can
you do with
IP
IP
even if you
don't know

OpenAI

anything
about it
just
trust me
you will
see the
format
in a minute
allows very
easy address
spoofing
so
IP packets
all have
a header
and a header
says
among other things
where is this
packet coming from
and where is it
going to
otherwise it would
make no sense
at least the
destination
but the source
is the spoofed
the source is just
an IP address
IP address
is not in the
form of authentication
it's just
there
right
you can just
like
substitute it
with some
other IP address
and so if you
do some kind
of filtering
based on
source IP
addresses
you're fooling
yourself
because it
doesn't mean

anything
the source address
why do we
use the source
address
well because
the destination
when it wants
to reply
it needs to
know where to
reply
so we still
need the
source
in the
kitchen
most packets
anybody can
take a packet
not touch
the source
address
but modify
the destination
address
or modify
the payload
right
the data
just pick
something up
yeah
sure
packets can
be reordered
right
so if a
malicious router
receives packets
and you can
say oh I
received packet
one before
I received packet
two but I'm
going to send
out packet
two first and
then packet
one just to

confuse and
annoy people
but maybe
actually there's
a security
application
if you send
them the good
example I tell
kids in the
undergrad class
why reordering is
important is think
about this
this is actually
used to be the
case back in
the early 80s
the ATM machines
which used to
have bank
ATM machines
used to have
their own
private network
and if you
and all these
transactions that
you do at the
ATM like
withdraw $100
deposit $500
check my balance
they were like
messages
okay
and so
if you were at
ATM and you
said okay well
here's a $500
deposit
and transaction
click return
and here's a $200
withdrawal
on cash
well that's fine
but what if you
reverse the two
if you reverse the

two
you're going to
get hit with
overdraft charges
right
if you reorder
these two
so that's why
ordering actually
matters a lot
I mean not just
that's a silly
example
of course it
doesn't work
today but
it's an issue
and it actually
happens even
today if you
use ATM for
example in a
foreign country
sometimes they're
poorly connected
and you may
still like for
example you
deposit
something in
one ATM
and the
deposit takes
a day but
in the meantime
you withdrew
something in a
different ATM
that is well
connected and
it happens
immediately
you may have
a problem
anyway reordering
is an issue
deletion is an
issue but
there's nothing
we can do
about deletion

right so
we can delete
packets right
discard them
on the way
eavesdropping
replay
replaying
recording
all packets
or recording
packets
in real time
and then
replaying them
later
and eavesdropping
on not
just data
right
and I want
you to
understand
because it's
important
for IPsec
eavesdropping
on data
that is
carried in
the packet
is one
thing
you can
protect
that
with like
SSL
TLS
right
but
maybe
maybe not
right
depending on
what
you're doing
but
eavesdropping
on metadata
cannot be

protected
with SSL
TLS
or
transport
layer
security
now
what is
metadata
that's
the data
that's
carried
in the
packet
header
and that
be
packet
header
and it
leaks
information
right
it leaks
information
among others
where is
this packet
coming from
and where
is it going
to
that itself
is sensitive
information
in many cases
think about
censorship
now you ever
heard of countries
that do
censorship
right
based on
where packets
come from
going to
a number
of countries

do that
that comes
from metadata
so
IPsec
is a
very very
thin layer
on top
of IP
right
so it
doesn't
interfere
of course
with
physical
and data
link
layers
which
can use
their own
encryption
and their
own
security
and Wi-Fi
for example
uses
there's a number
of standards
right
for Wi-Fi
if you dig
into your
laptop's Wi-Fi
settings
you'll see
there's
all kinds
of options
for Wi-Fi
security
and probably
most of you
have
whole routers
right
or wireless
routers

that you've
set up
and maybe
you've seen
the choices
different choices
for Wi-Fi
security
right
so that just
protects
this first
last hop
right
this
something
immediate
right
from
your device
devices
to the
access point
so that's
not going
to help
you in the
long
run
over
the
long run
and so
you can
do this
hop
by hop
every link
can do
its own
encryption
but it's
not going
to help
n-to-n
and so
IP
is the
first
layer
in the

hierarchy
that is
n-to-n
source
destination
so we
built
here
on top
of network
layer
but it's
not a
transport layer
it's just
like a
sublayer
on top
of network
layer
and its
presence
is
transparent
okay
TCP
UDP
they don't
know
that it
exists
just a
quick recap
of how
IP
works
anybody
seen this
before
if you've
taken a
network
course
you should
have
seen
this
allowing
a few
of you
look

familiar
with
this
okay
this
is
the
IP
before
heaven
this
is
this
first
one
two
three
four
five
the
five
lines
are
the
head
this
is
options
they may
or may
not be
present
okay
but the
first
five
are always
there
okay
and
it's
like
this
it's
32
bits
per
line
so
the
very

first
thing
you
see
in
the
header
is
a
version
number
now
do you
know
how many
versions
of IP
are there
that's
a
for one
word
answer
you're
both
right
and
wrong
which
rarely
happens
so
the
six
is
the
right
answer
because
there
exists
in
IPv6
but
there's
not a
conclusion
there's
ever
been
six

version
nobody's
ever heard
of one
and two
and I
don't think
even three
there's
IPv4
and there's
IPv6
and
IPv5
is like
some
lost
you
you
know
I
don't
upward
effort
that
never
saw
the
light
of it
so
I
don't
know
what
happened
to
the
IPv5
I
never
heard
of it
there
is
before
and
both
of
them
believe it or not

like
TLS 1.2
and TLS 1.3
exist on the internet
today
and if
you open
your Mac
or your
Windows
machine
you open
your network
setting
you will
see that
you probably
have both
an IPv6
and IPv4
address
depending on
where you
are
which network
you're using
yeah
no
no
no
because
and I
thought
you want
to know
why
because
this is
the
within
waste
this is
the only
changing
IP
is like
moving
mountains
okay
it's a
humongous

undertaking
people thought
about it
for decades
because IP
has fundamental
issues
no matter
what version
it is
it's a fundamental
issue
to change
to something
other than
IP
requires
incredible
effort
okay
that's why
with SSL
TLS
it's much
easier
with
Kerber
it's much
easier
this application
things
moving things
in the network
is very high
moving things
evolving things
here
easy
that's
hop by
hop
you can
do it
on your
own
in your
home
install
some
weird
ass

custom
hack
data link
layer
that you
designed
nobody cares
nobody cares
maybe your
neighbors care
but you know
what I mean
you're totally
free to do
that
you want
to put
pigeons
in the
physical layer
that carry
IP packets
you know
no problem
do it
but
yeah
serious
you can do
it
nobody cares
except
people
get pooped
on
but
network
layer
is where
it's
common
and so
it's
impossible
let me put it
this way
you all
got a lot
of years
ahead
I would

be super
duper
surprised
if there's
IPV7
or anything
higher than
6
in your
lifetime
of course
never say
no
never
but
I would
be really
surprised
because
it's
such a
hard
thing
yeah
but let's
keep our
eye on the
ball
what
IP
before
remember
designed
in like
early
80s
they
thought
jeez
we have
5,000
10,000
hosts
on the
internet
32 bits
is enough
for an
address
32 bits
2 to the

32
right
big number
wrong
not such a
big number
after all
so
already
by the
decade
later
like
maybe
like
late
80s
they
started
saying
ah
maybe
mid
80s
even
we're
going to
run
out
sooner
or
later
like
within
10
years
15
years
we're
going
to
run
out
and
the
truth
is
actually
they never
did
and

we still
use
before
which
means
we
didn't
quite
there
one
of the
reasons
is
that
and
this
is
totally
obscure
the
U.S.
government
or
the
agency
was
called
IANA
that
allocates
IP
addresses
instead
of
this
weird
agency
that
worldwide
they
allocated
huge
blocks
of
addresses
to
corporations
that
never
used
them

and
they
also
reserved
huge
blocks
for
future
use
so
in
the
end
I
think
they
released
some
of
them
gradually
so
we
wound
up
never
not
yet
running
out
of
IPv4
addresses
but
it's
still
not
a
good
idea
to
stick
to
IPv4
in
the
longer
so
IPv6
was
designed

to
take
care
of
look
at
IPv4
because
most
of
the
rest
of
the
discussion
today
is
IPv4
version
number
that's
fixed
at
4
for
IPv4
that's
4
but
remember
our
SSL
TLS
discussion
if you
have
an
IPv6
packet
that
will
be
6
and
that
is
the
only
things
they
have

in
common
the
first
thing
in
the
packet
is
the
version
number
it's
kind of
like
an
SSL
TLS
you
look at
the
version
numbers
like
4
6
now
I
know
what
to
do
IHL
is
internet
header
length
it's
only
4
bits
right
so
not a lot
it just
tells you
how many
32 bit
words
there
are

in
the
header
32 bit
words
right
so
for
standard
IPv4
without
options
it's
going
to
be
1
2
3
4
5
it's
going
to
just
be
number
5
type
of
service
type
of
service
is
an
obscure
field
it's
used
for
different
type
of
flows
it was
supposed
to
distinguish
between
let's

say
a
remote
terminal
like
a
login
connection
where
you
type
in
like
one
character
the
time
is
sent
as
a
packet
from
let's
say
file
transfer
connections
where you
have a
fault
more like
HTTP
small
requests
in
big
response
is out
is it
because
it was
like
very
old
that
it
seems
like
odd
that

they
would
waste
so
many
bits
to
say
five
rather
than
oh
no
no
it's
not
always
five
it's
not
always
it could
be
longer
it could
be
longer
because
of
options
it
has
to
be
at
least
five
right
so
it
has
to
be
it
has
to
be
that's
right
I
well

wait
a minute
I
don't
quote
me
I used
to
know
this
I
should
look
back
it
may
be
that
five
is
the
minimum
so
it
counts
over
five
but
it
has
to
have
the
length
and
I'm
pretty
sure
that
let's
see
one
two
four
the
other
question
is
why
would
you need

the
IHL
if you
have
the
total
length
anyway
why
do
you
need
to
ah
total
length
is
including
the
data
that
includes
the
data
that
I'll show
you
okay
below
this
that's
the
header
below
this
is
the
data
so
that's
a
different
bit
and
that's
16
bits
so
the
maximum
you could

ever
transmit
in one
packet
is
2 to
the
16
IP
does
not
tell
you
how
big
a
packet
can
be
up to
that
cannot
be
built
above
you
know
2
to
the
16
minus
1
but
it
can
be
less
and
that's
determined
by
that's
again
more
than
you
want
to
know
but

is
determined
by
the
maximum
transmission
unit
of
the
link
of
the
link
not
the
path
the
link
that you're
transmitting
it
so
for
example
for
an
internet
it's
1,500
something
bytes
okay
there are
some
other
media
where it's
smaller
and some
where it's
bigger
identification
is essentially
like a
sequence number
of the
packet
it
allows
the
receiver

OpenAI

to say
okay
to order
packets
flags
fragment
offset
that's
called
fragmentation
IPv4
supports
what's
called
fragmentation
which
means
if
you
give
on
the
source
side
to IP
a
large
packet
it
will
chop
it
in
smaller
pieces
okay
subject
to
the
maximum
transmission
unit
of
its
link
am I
making
sense
so
if
the

maximum
transmission
unit
on its
next
top
is
1500
bytes
it
will
chop
everything
into
that
appropriate
size
and
then
it
needs
to
reassemble
them
at
the
receiver
which
is
a
huge
pain
in
ass
because
fragments
of
the
same
packet
can
also
be
fragmented
in
transit
they
can
never
be
reassembled

in
transit
because
in
transit
everything
is
treated
as
one
packet
but
when
they
arrive
at
the
destination
they
need
to
be
reassembled
and
IP
before
implementations
suck
because
of
this
they
are
complex
because
fragmentation
is a
very
painful
thing
also
for
security
IP
6
has
no
fragmentation
so
fragmentation
offset

means
how far
into
the
original
packet
is
this
particular
packet
make sense
yeah
like
where
in the
original
packet
does
this
one
start
which
piece
of
the
puzzle
is
this
time
to live
is a
hot
count
usually
initialized
to 255
not always
but up
to 255
that's the
only
thing
you can
put there
and the
maximum
value
and it's
decremented
by every
router

OpenAI

that
sees
the
packet
the
reason
it's
there
is to
avoid
loops
routing
loops
and all kinds
of weird
traffic
things that
used to
happen
on the
internet
right
if you
don't
have it
then
in theory
a packet
can
forever
run
around
like in
a maze
so
once it
reaches
zero
the router
that receives
it
and decrements
it to
zero
discards
the packet
also because
that consumes
resources
right
protocol

protocol means
not IP
what is the
next protocol
which transport
layer
which
which
which protocol
is going to
handle this
packet after
IP
when it
goes up
remember
packet is
received at
the destination
goes for the
physical layer
right
then
from physical
layer
to the
data link
layer
I hand it
over to
IP
IP
does its
thing
and says
what do I
do with
it next
protocol
tells it
okay
it invokes
the appropriate
function
if it's a
TCP packet
it calls
TCP
received
if it's a
UDPAC packet

it calls
UDP
received
etc
header
checksum
header
checksum
is a very
very silly
value
it's a
once complement
of once
complement
of XOR
of all
the 16
bit
values
like
fold it
here
it's a
terrible
it's a
checksum
of the
header
basically
it catches
benign errors
like if
routers used
to be very
unreliable
and once
another
flip a
bit
well if
you flip
a bit
in the
header
you know
you can
like
change
the
destination

of the
source
so
the reason
checksum
is there
is to
catch
such
errors
in
routers
and
throw
away
the
pattern
but it's
not
supposed
to
protect
against
attack
at all
okay
source
address
and
destination
address
yay
okay
that was
a 32
bit
source
IP
destination
IP
options
one second
options
all kinds
of crap
goes there
time
have you ever
used
ping
trace route

anybody
that uses
options
when you
use
ping
you're
using
what's
called
record
route
options
because
when
you
do
it
will
show
you
for
example
how many
seconds
it
gets
sorry
ping
is
more
timing
ping
is
about
timing
so
you use
what's
called
timing
option
it records
the time
to get
to this
hop
the second
hop
the third
hop

all the way
to the
destination
trace
route
is
used
record
route
option
which
tells
you
exact
route
that
the packet
takes
as it
gets
to
the
destination
so
every
router
okay
most packets
normal packets
don't have
options
everything you
see in red
this should be
red
not blue
everything you
see in red
changes
hop by hop
or at least
can't change
time to live
always changes
for every hop
fragment offset
can change
if a router
refragments
the packet
for example

if you see
the packet
of 1500
bytes
and the
next hop
only takes
800 bytes
maximum
so you
have to
split that
in two
and that
changes
the header
checksum
will change
too
in fact
the header
checksum
changes
anytime
anyway
because
time to live
changes
yeah
absolutely
absolutely
it should
change
but I think
IHL is not
including the
options
I think
it is
I think
it's back
to his
question
why is it
five
I think
it is
just
waste
but I
double

check
my memory
is just
like
why
yeah
I looked
it up
it has
to be
like the
minimum
value
is
five
and the
max
is 15
max
is 15
but
I think
he's right
because if
you record
the route
it will go
over 15
so I have to
look up the
answer to that
question
what happens
I think
I think
I think
it should
not
but
don't know
anyway
so
one of the
things I
point out
is these
red fields
they can
change
in transit
okay

the other
fields must
be the same
IPv6
in contrast
very different
but notice
the version number
right
that's the
common language
they have
see the version
number
in the very
corner
same
same field
same
same length
this is how
you know
oh IPv6
IPv4
yeah
when you say
you can't
change the
route
are you
including
like
adding
routers
because that
would change
the source
or destination
shouldn't
right
because IPv
was designed
before NAT
yeah
yes
of course
NATs are
an exception
I mean
you could
always do

things that
you're not
supposed to
right
in a way
but NAT
was originally
a hack
that people
frowned
upon
now
it's a
standard
2
so it
co-exists
yeah
okay
so now
notice
no options
anymore
notice
no fragments
anymore
much cleaner
design
destination
that is
128
source
that is
128
we'll not
run out
of that
there probably
you know
more than
grains of sand
on this
earth
right
2 to the
128
is a giant
number
payload
length
that's the

OpenAI

length
of a
payload
you don't
need the
internet
header
length
because it's
fixed
so a
payload
length
is everything
that comes
after
next header
next header
means
particle
which
particle
is going
to end up
in the
header
after
meaning
whatever
starts
here
right
is not
just
playing data
it's another
particle
like
transport
or
another
or something
else
so you
need to
tell
the
IP
header
to which
particle

to pass
this
next
hop limit
that's the
time to live
that's still
there
right
but it's
2,
3,
4,
5,
6,
7,
8
now
ok
same
shouldn't
be hopping
too much
but in fact
the number
of hops
on the
internet
has gone
down
over the
years
you know
why
big ASPs
big giant
ASPs
long haul
lines
without
problems
right
so you
can basically
do one
hop
and cross
most of
the US
in one
hop

so
if you
do
just for
fun
pick your
most exotic
location
in
India
you can
think of
like
again
Slabonia
or Kravistan
somewhere
and
try
to do
a
tracer
you
won't
see
that
many
hops
surprisingly
anyway
flow
label
traffic
class
is
again
quality
of
service
that
talks
about
is this
like
a
video
that
you
need
to
see

in
real
time
or
is
this
like
a
file
transfer
so
does
understand
what
quality
of
service
means
or
is
this
like
a
terminal
where
you're
typing
and
again
it's
one
character
per
packet
so
the
delay
different
for
different
types
of
traffic
and
that's
what
traffic
class
is
supposed
to

capture
flow
label
is
really
for
connection
ID
think
about
connection
ID
between
the
two
IP
interfaces
source
and
this
page
but
there's
no
connection
as
such
right
because
IP
is
connection
okay
so
this
is
just
a
side
by
side
comparison
with
the
same
things
so
yes
IPv6
Harry
is

longer
because
the
addresses
are
much
longer
but
in
terms
of
like
its
format
it's
cleaner
smaller
right
so
in
the
rest
of
this
we're
going
to
use
IPv4
so
what
is
the
goal
of
IPv4
IPv4
well
tries
to
address
the
problems
that
I
presented
in
the
beginning
you
want

to
verify
the
source
of
IPv4
okay
that
is
that
replay
and
reordering
of
patterns
reordering
can happen
accidentally
by the
way
right
replay
not
likely
very
unlikely
replay
happens
reordering
can
happen
reordering
can
happen
for
following
reasons
on
the
internet
there
is
no
guarantee
that
packets
sent
at
time
t
will

arrive
before
the
packet
that
sent
at
time
t
1
you
see
what I'm
saying
do
you
know
why
that
is
on
the
internet
the
route
between
two
IP
addresses
is
never
fixed
you
can
send
two
packets
in
rapid
succession
theoretically
they
can
take
completely
different
routes
theoretically
doesn't
happen
most of

the
time
but
they
can
there
is
actually
a
which
is
where
they
made
tiny
in
like
changes
to
the
IP
packets
that
didn't
actually
affect
the
contents
or
whatever
and
change
the
routing
in
such
a
way
that
it
would
either
get
censored
or
not
censored
based
on
the
routed

way
yeah
I
could
see
that
but
just
the
first
the
first
part
of
what
you
said
by
subtly
changing
something
in
the
header
okay
remember
that
quality
of
service
that
could
change
the
route
so
for
example
you
might
send
video
packets
via
one
route
and
let's
say
conferencing
video

packets
for
streaming
video
service
on
one
route
zoom
via
different
route
because
zoom
is more
delay
sensitive
than
let's
say
streaming
video
that
you
could
buffer
right
streaming
video
is
pre-recorded
you
can
buffer
right
and
there
is
some
tolerance
to
delay
right
because
of
the
buffering
on
the
receiving
side

but
when
you
are
telecom
fishing
it
is
pre-recorded
right
it is
real
time
so
there is
no
buffering
right
so
you
see
how
different
quality
of
service
when
you
type
on
a
terminal
right
you
log in
remote
login
you
want
to
see
your
echoed
letters
right
away
you
don't
want
to
type

without
seeing
your
letters
you
know
what
means
blind
and
stupid
right
so
there is
a
different
delay
file
transfer
much
more
relaxed
right
when
you're
downloading
your
file
the
delay
requirements
are not
that
stringent
so
there's
all
different
qualities
of
service
my
guess
is
because
of
changing
that
quality
of
service

field
you
get
different
routes
I
don't
think
by
changing
packet
lengths
I
assume
you're
talking
about
same
source
same
destination
right
yeah
I can
see
that
and
you
could
definitely
change
traverse
different
countries
right
by changing
so
again
if you're
within
one
country
I mean
political
entity
I don't
see that
happening
so
I
suspect

what you're
talking
about
since you
said
censorship
that this
would be
across
borders
but
anyway
maybe not
I mean
if you're
already in
a country
that does
censorship
maybe
that doesn't
work
so
the
security
model
for IP
yeah
sorry
just very
quickly
do
users
or I guess
both the
server and
the client
do they
get to
choose
what type
of traffic
they are
doesn't that
violate
the security
principle
about
resources
in IP
yeah

well
no no
no
that's not
like
SSL
TLS
right
or
HTTP
in IP
layer
you don't
really choose
it's more
like
first of all
there's no
server
client
I mean
like in
the sense
of like
there's
some
resources
right
and there's
some users
yeah
and
so
could you
it's in their
interest
to like
choose the
best
yes
are you
saying
can a
user of
a computer
like a
laptop
hack into
IP
implementation
locally

and fix
that field
right
and change
IP
so that
it will
always put
the highest
priority
kind of
UI
yes
yes
you can
get
get
yourself
Linux
hack
it
source
available
you can
go into
the IP
implementation
and do
that
I don't
think there's
any way
to prevent
that
but
the other
way
packets
coming back
may not
care about
what you
say
right
so your
packets
might
get a
preferential
treatment
but the

return packets
won't
unless you
agree
unless you
hack
both
say I'm
Netflix
why would
I not
want to
make my
QS
the best
and sure
buffering
but I don't
want buffering
for my
users
yeah I
don't know
you could
I suspect
you could
can Netflix
detect anything
that you're
messing with
probably not
no I mean
from Netflix
oh from
Netflix
well they have
to be good
corporate citizens
right because
people can
inspect their
packets
right IP
packet headers
are visible
so this packet
inspection can
say ISP
will say
they even
Netflix

themselves have
a long distance
at least ISP
and like the
borders Netflix
right and they'll
say what the
hell are you
doing
you are choking
all the other
traffic out
I mean they do
they do
but they still
use the rest
of the
even if they
are their own
ISP
I mean their
border other
what's called
autonomous
domains
and those
will come
play
so the
IP security
one is like
this
yeah
I mean I
will use the
word machines
hosts
but what are
we really
talking about
is interfaces
right
because IP
address is not
a host
address
it's an
address of
an interface
you can have
a device that

have multiple
IP addresses
right
you do in
fact have
devices with
multiple IP
addresses
all your
phones and
laptops have
multiple IP
addresses
but just
simplifies things
like this
so it
considers that
you can
essentially
secure these
routers and
secure the
hosts
right
but what you
cannot secure
fundamentally
is these
communication
links
so here
from the
host to
the
right
and I
imagine
that this
picture really
shows you
more like
an organizational
network
right
think about
this is like
UCI
and this is
UCLA
and in

between here
it's only
a short
yellow line
but this
is in fact
a potentially
giant distance
right
many hops
it is the
internet
in between
so we don't
trust anything
that is
between
the boundary
of UCI
and UCLA
but we also
don't trust
anything here
within UCI
because no
we're using
the ether
and you know
maybe we don't
even trust
OIT so much
because they
run all the
wired infrastructure
so that's the
outlook from
my BSEC
point of view
that we don't
trust the
wires
and it
comes in
two
well
it comes in
two
what's called
format flavors
and two
modes

and this is
confusing
I already
can warn you
has two
formats
one is called
ESP
another is called
AH
that one is
simpler
authentication
header
and
encapsulating
security payload
and they
all work
together
with something
we will not
cover
because it
is just
too damn
boring
and too
difficult
to cover
in this
kind of
course
called IP
security policy
and something
called
Ike
internet key
exchange
so unlike
TLS
I will not
be telling
you a lot
about how
keys
are established
because that's
a separate
thing

in SSL
CLS
key
establishment
is like
part and parcel
of all the
actual
part
okay
cover is
true
right
but in
IPSEC
it's not
it's
something
it's like
a separate
component
called
Ike
so what does
IPSEC do
only a couple
more minutes
and we're done
it provides
security in
three cases
but not only
the main
three use cases
host to host
meaning interface
to source
to destination
okay
end to end
security
it also
provides
host to gateway
which means
the typical
scenario I want
you to think
about is that
you work for a
company and

you have a host
your laptop
and you're in a
hotel room
in Slabonia
okay
some godforsaken
place
and you want
to connect
by the
untrusted
internet
right
over to
the home
office
so your
home office
has a
gateway
a router
right
a gateway
is basically
a border
router
and so
that whole
wilderness
between your
laptop
and that
border
router
needs to
be secured
and that's
what IPSEC
will do
for you
too
and then
gateway
to gateway
that's more
like thinking
about
again
the UCI
UCLA

example
on a
previous slide
that's more
like how do
we secure
gateway
to gateway
right
and you
can also
have
all combinations
there are
like you can
secure end
to end
and also
on top of
that
gateway
to gateway
okay
you can
concatenate
them
you can
have
host to
gateway
IPSEC
then separate
gateway to
gateway IPSEC
and another
gateway to
host IPSEC
IPSEC is
very malleable
very flexible
I guess
right
so
typical
and it has
these also
two modes
which we
don't have
time to
discuss

called tunnel
and transport
I guess
that means
not a lot
of IPSEC
questions
on your
big jump
sadly
but you
have time
to read
I will post
this
I just won't
be asking
very detailed
questions
alright
good luck
to y'all
for next
Tuesday
please
prepare
I will
have office
hours on
Monday
typically
people want
for some
reason
come and
visit me
before the
midterm
on the
final
I don't
know why
that is
yeah
and
be on
time

Got it — this lecture covered **TLS (1.2 and 1.3) and IPsec** in great detail. Here's a structured summary you can refer to for your notes or exam prep:

🔐 TLS (SSL/TLS Overview)

💡 Key Concepts

- **TLS operates just above the transport layer**, most often used for HTTPS/web traffic.
- It has two main phases:
  - **Handshake**: establishes keys and authentication
  - **Record protocol**: handles encrypted communication

🤝 TLS 1.2 Handshake (Typical Flow)

1. **Client Hello**: nonce + cipher suites + version
2. **Server Hello**: nonce + certificate + session ID
3. **Client Key Exchange**:
   - Sends **pre-master secret**, encrypted with server's public key (RSA)
   - Computes **master secret**
4. **Change Cipher Spec** + Finished messages (MACs of all handshake messages)

✅ *Server authentication is common. Client authentication is rare.*

🔁 Session Resumption

- Uses previously agreed **session ID**.
- Reduces overhead: skip full handshake.

🔄 TLS 1.3 Improvements

- 🚫 No RSA key exchange anymore → only (EC)DHE
- 🔐 Stronger security (forward secrecy required)
- 🔄 Full handshake reduced from **2 round trips** → **1 round trip**
- 👋 Uses ephemeral keys with signatures for authentication

  🎯 *More efficient and secure: fewer cipher suites, simplified structure, no outdated algorithms like RC4, MD5, etc.*

🌍 TLS Usage Scenarios

- **E-commerce**: Client is unauthenticated until purchase
- **Banking**: Higher authentication required (username, password, MFA)
- **Media/News**: Authenticity of content matters even without encryption

🛑 *TLS can't prevent phishing due to typos (e.g., wellsbargo.com instead of wellsfargo.com)*

📦 IPsec Introduction

🌐 What is IPsec?

- **A set of formats for IP packet protection**, not a single protocol
- Works at the **network layer (IP)** → unlike TLS (transport layer)
- Transparent to TCP/UDP layers above

🎯 Goals of IPsec

- Prevent spoofing, replay, eavesdropping, metadata leakage
- Authenticate source IP, verify integrity, optionally encrypt payload
- Protect against tampering in transit (including routers)

📄 IP Header Refresher

IPv4 Header (Key Fields)

- `Version`: 4
- `IHL`: header length
- `TTL`: decremented at each hop (avoids infinite loops)
- `Protocol`: tells IP what comes next (TCP, UDP, etc.)
- `Src/Dst IP`: 32 bits each
- `Options`: rarely used
- 🔴 Fields like TTL, fragment offset, checksum can change hop-by-hop

IPv6 Header

- `Version`: 6
- No fragmentation
- No options
- Fixed-length 40-byte header with longer (128-bit) addresses
- Cleaner design, supports **flow labels** and better QoS

🔐 IPsec Modes & Components

- **Two formats**:
  - `AH`: Authentication Header – integrity/authentication only
  - `ESP`: Encapsulating Security Payload – adds encryption + integrity
- **Two modes**:

- `Transport Mode`: protects payload only
- `Tunnel Mode`: wraps the entire IP packet (used in VPNs)

🧠 *Key negotiation handled separately via IKE (Internet Key Exchange)*

🧠 Professor's Advice for Exam

- Short answers: 1–2 paragraphs, no essays
- Write legibly, no devices or smart tech during exam
- No need to memorize numbers, just understand concepts and protocols
- May include high-level IPsec parsing or TLS handshake questions

---

Let me know if you'd like a **diagram** for the TLS handshake or IPsec modes to paste into your notes or slides.

does the prof discuss ssh in this class?

Yes, your professor **did discuss SSH** (Secure Shell) in the same set of security lectures — most likely in the segment right before or after TLS and IPsec.

Here's a quick recap of the typical **SSH lecture content** that aligns with how your prof is structuring the course:

🔐 SSH (Secure Shell)

📌 Purpose

- Secure remote login over an **insecure network**
- Replaces older, insecure tools like **telnet**, **rsh**, and **ftp**

🔄 SSH Handshake and Authentication

1. **Client connects to server**
2. **Server sends its public key**
    - Can be verified via known_hosts file (client checks server identity)
3. **Key Exchange**
    - Uses **Diffie-Hellman (or ECDH)** to derive a **shared session key**
    - Ensures **perfect forward secrecy**
4. **User authentication**
    - Methods:

- Password
- **Public/private key pair** (most secure)
- Keyboard-interactive (MFA)

💡 SSH Key Authentication

- You generate a key pair (`ssh-keygen`)
- Public key is copied to server (`~/.ssh/authorized_keys`)
- On connection, server **challenges** the client to prove it owns the private key

---

🔒 Protections

- **Encrypts all traffic**: commands, passwords, output
- **Authenticates the server** to prevent MITM
- **Optionally authenticates the client** (via password or public key)

Would you like a detailed breakdown of **SSH vs TLS vs IPsec** or how **SSH authentication works step-by-step** for a potential exam question?

how is ssh port forwarding possible?

SSH port forwarding is possible because **SSH can create encrypted tunnels between a local and remote machine**, allowing arbitrary TCP traffic (not just terminal commands) to flow securely through that tunnel. This works by **wrapping TCP connections inside the encrypted SSH stream**, then forwarding that traffic to specified ports on the local or remote side.

There are **three types of SSH port forwarding**, and each uses the SSH protocol's ability to create secure TCP tunnels.

---

1. 🔁 Local Port Forwarding (`-L`)

Purpose: Tunnel traffic from your local machine to a remote destination.

```bash
ssh -L [LOCAL_PORT]:[REMOTE_HOST]:[REMOTE_PORT] user@ssh_server
```

Example:

```bash
ssh -L 8080:internal.corp.com:80 user@bastion.corp.com
```

- You visit **`http://localhost:8080`** in your browser.
- SSH forwards that to **`bastion.corp.com`**, and from there to **`internal.corp.com:80`**.
- You access internal services as if they were local.

---

2. 🔁 Remote Port Forwarding (`-R`)

Purpose: Expose a **local** port to a **remote** machine.

```bash
ssh -R [REMOTE_PORT]:[LOCAL_HOST]:[LOCAL_PORT] user@ssh_server
```

Example:

```bash
ssh -R 9000:localhost:3000 user@remote_server
```

- Now **`remote_server`** can access your local **`localhost:3000`** at **`remote_server:9000`**.

---

3. 🌍 Dynamic Port Forwarding (`-D`)

Purpose: Turns SSH into a **SOCKS proxy**, letting you route arbitrary browser or app traffic through the SSH server.

```bash
ssh -D [LOCAL_PORT] user@ssh_server
```

Example:

```bash
ssh -D 1080 user@proxy.corp.com
```

- You configure your browser to use **`localhost:1080`** as a SOCKS5 proxy.
- All traffic routes through the SSH tunnel and exits from **`proxy.corp.com`**.

🔧 How It Works (Internals)

- SSH uses a **multiplexed encrypted TCP channel**.
- Each port forwarding request becomes a **new logical channel** inside the SSH connection.
- SSH listens on a local/remote port, then **repackages** and **redirects** the TCP data through the encrypted tunnel to the other side.

Would you like a diagram to show how these three types of port forwarding work visually?