

IIT Madras
ONLINE DEGREE

Modern Application Development – I
Professor Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology Madras
HTTPS

Hello, everyone, and welcome to this course on modern application development.

(Refer Slide Time: 00:17)

HTTPS

Now let us look at this concept of HTTPS. The so called secure version of HTTP.

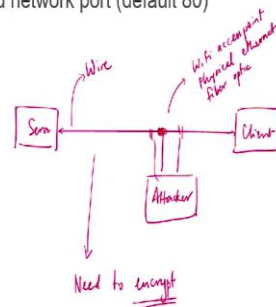
(Refer Slide Time: 00:23)

Normal HTTP process

- Open connection to server on fixed network port (default 80)
- Transmit HTTP request
- Receive HTTP response

Safety of transmitted data?

- Can be tapped
- Can be altered!



In order to understand what HTTPS is and why it is required, let us first understand what happens in a normal HTTP process. The client opens a connection to the server, or a fixed

network port, typically port 80. It transmits an HTTP request, which we already know what it looks like, it is object/HTTP/1.1, or 1.0, or whatever the version of... and there may be some additional headers.

But all of those are things which anybody can read them, and they are essentially English language headers. It also gets back HTTP response, which once again, has a bunch of headers, which are in English. And some body text, which is whatever needs to be sent back. Now the problem over here is, let us say that I have a server and a client, they are connected well by some kind of link, I keep saying the wire.

But in practice, it could be multiple things, you might have a WiFi connection to your router from the router, it has, let us say, an internet connection to the nearest switch, from the switch to the local exchange, it could be a fiber optic connection. Then, then there are like multiple fiber links that it goes through. And finally, it reaches the server. All of that put together is abstracted out into one wire.

And I could potentially think of someone sitting out here. Who is an attacker, when I say an attacker, it is not someone coming to beat you, but it is someone who is trying to get information about you that you do not want them to have? So, the attacker is basically sitting over here, and has somehow managed to tap into the wire. So, what could this be? It could be your WiFi access point.

Maybe you are in a coffee shop somewhere Starbucks or whatever it is, and you have turned on your Wi Fi and you are connecting with the local Wi Fi access point without authentication. Or it could be physical Ethernet. They managed to go in there and tap a couple of wires that are actually present physically. Or it could be the fiber optic itself, they managed to go and actually split the fiber optic cable put in a splitter, and collect the data that is going on it.

So, you do not really get to see the that do not understand that there is a break in the cable, because they are still passing the information through. But they also have a splitter that allows them to see what information is going on with cable. So, all these things are possible. And I want to make sure that the information that I send to the server cannot be read and understood by the attacker.

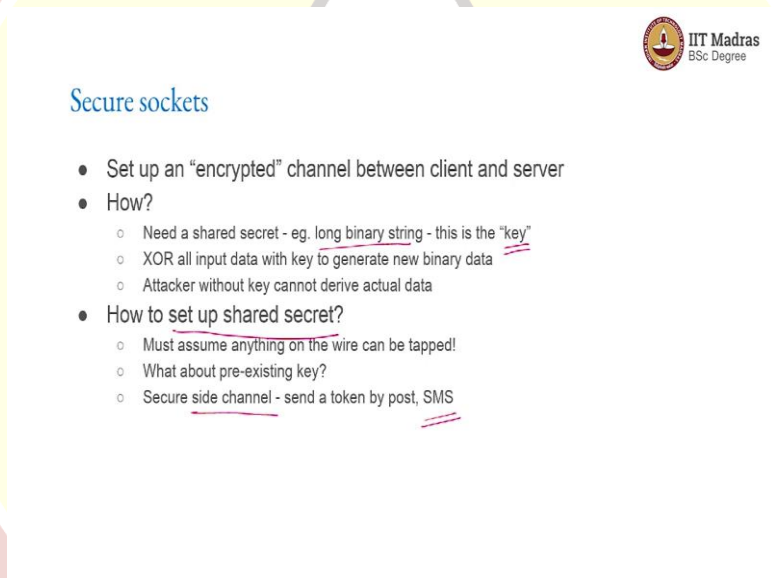
That is the best that I can say. I cannot prevent them from actually seeing the 1s and 0s on the line. But as long as those 1s and 0s do not make any sense to them, I am probably safe. So,

what it means, in other words is that this wire, I need to encrypt it. Encrypt, meaning I need to use some kind of a mathematical function, which will convert it into a different set of bits, which I will understand, the server understands, but nobody else can making sense of.

And one other thing that could potentially be done is it is not just about getting the information. If they actually have a tab or there, they could potentially even alter the information. So, they could take a get request from you and change it into let us say a post request that changes your email ID or something of that sort.

Because, after all they could also just click this somewhere and take a link to themselves, in order to change the information that is being sent. So, all of that is possible, which means that there has to be some way by which I can guarantee that wire cannot be tapped or altered.

(Refer Slide Time: 04:32)



Secure sockets

- Set up an “encrypted” channel between client and server
- How?
 - Need a shared secret - eg. long binary string - this is the “key”
 - XOR all input data with key to generate new binary data
 - Attacker without key cannot derive actual data
- How to set up shared secret?
 - Must assume anything on the wire can be tapped!
 - What about pre-existing key?
 - Secure side channel - send a token by post, SMS

Now for that, essentially there is this concept of the Secure Sockets Layer. The secure socket layer essentially sets up an encrypted channel between the client and the server. How do we do it? I need to have some kind of a shared secret between the client and the server. So, typically, this is some kind of a long binary string, some big number.

This is the so called encryption key. And once I have such a screen, I can basically take whatever data I am planning to send, and just use the XOR function, or some kind of mathematical function to combine my key with the data to be transmitted, the simplest form, that is all that you need to do. And the point is, because this long binary string was unknown to the attacker, they cannot invert it and get back the original information I was going to send.

So, the attacker without the key cannot derive the actual data. And that is all that I really care about. It also means, that if they do not have the key, and they try splitting the line, and pretending to be me, that also will not work, because they need that key in order to create the new request to be sent to the server. Otherwise, the server will say that the key was invalid, I do not understand what request you are trying to send over and reject it.

So, it all comes down to how do you set up a shared secret? You must assume that anything that you are talking to the server can be tapped. In other words, it is like you are sitting on one end of a room, the server is somebody who is sitting on the other end of the room, and you are trying to talk to each other. And you need to establish some kind of code words between yourselves, but everybody else in the room can also listen to you.

So, how do you do this? You cannot really do it easily, unless there was some other piece of information, that both of you trust, and can be used in order to set up this kind of shared secret. Effectively, it is creating something called a side channel. I can sort of say like, okay, look, my partner at the other end, also knows some other third person or know something about them.

Then I can take some mathematical functions related to some common information that both of us have, create a key, which is related to something that only I have, send it across to the other person in a way that they can understand. And anybody else who is listening can also understand, but then they combine it with the other secret of their own and send it back to me. So, finally, a secret that I have, and a secret that the server has, has been combined in such a way that nobody else who is listening to this conversation will be able to find out the actual nature of the channel.

How exactly this is done is way beyond the scope of what we can do in this course. It is a fascinating topic, and anybody who is interested should really spend some time reading about it. But for the time being, let us just assume it can be done. And one way that you can think about it is, which you are familiar with, could even be that maybe I send a one time password on SMS, that might be one way by which I can share a piece of information too.

Obviously, the one time passwords are only like a few digits long. So, by themselves, they are not long enough to be used as encryption keys. But they can be used to derive an encryption on the other side.

(Refer Slide Time: 08:05)



Types of security

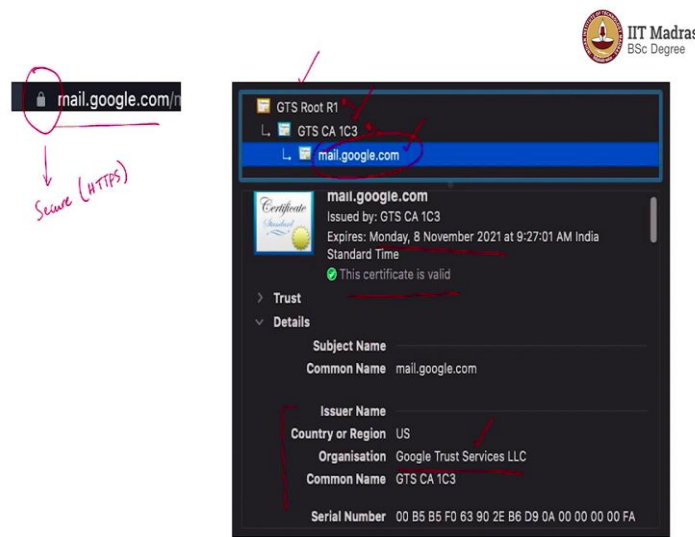
- Channel (wire) security
 - Ensure that no one can tap the channel - most basic need for other auth mechanisms etc.
- Server authentication
 - How do we know we are actually connecting to mail.google.com and not some other server?
 - DNS hijacking possible - redirect to another server!
 - Server certificates
 - Common root of trust needed - someone who "vouches for" mail.google.com
- Client certificate
 - Rare but useful - server can require client certificate
 - Used especially in corporate intranets etc.

So, like I said, there are a few different forms of security. One is the channel level security, that is the wire itself. And we want to ensure that no one can tap the channel. Now, in the process of doing this, I also need to be able to trust that the server at the other end that I am talking to is the correct server. It cannot be just someone who says, I am the Gmail server, connect to me, and give me your username and password, and I will show you all your (())(08:35).

I need to be very sure that the server at the other end is the correct server before I give the username and password. So, server authentication is important. And similarly, another form that is used to sometimes we use something called a client certificate. This is rare, but is actually a very good form of authentication, makes it in some ways, it is even better than using passwords.

Because it is, you cannot sort of distribute the client certificate using, you cannot just lose it. So, if you have a client certificate, then the server can rely on the client certificate to say, yes, I know that you are the right person. And here, you can have this information is meant for you.

(Refer Slide Time: 09:19)



Types of security

- Channel (wire) security
 - Ensure that no one can tap the channel - most basic need for other auth mechanisms etc.
- Server authentication
 - How do we know we are actually connecting to mail.google.com and not some other server?
 - DNS hijacking possible - redirect to another server!
 - Server certificates
 - Common root of trust needed - someone who "vouches for" mail.google.com
- Client certificate
 - Rare but useful - server can require client certificate
 - Used especially in corporate intranets etc.

So, how does this sort of, trusting that I am actually talking to Google happen? We have this notion of a common root of trust. And what I mean by that is, I personally do not know who runs the Gmail server. So, how do I sort of trust that, I am actually talking to the Gmail server? Well, how do you actually trust a person that you are meeting out for the first time?

Let us say you have a common friend who is able to introduce both of you. In that case, you can really say that, I trust this friend, you also trust this friend, so fine I will trust you. So, I trust that you are who you say you are, because this friend knows both of us and he is introducing us. In the same way there is this notion of certificates. So for example, in the browser bar, if you, when you type in the mail.google.com, you will see this padlock icon.

This basically indicates the security. This basically says HTTPS. And, you might find the different browsers show this in different colors to indicate that yes, it is secure, or it is actually, some kind of validation, secure some extra validation, and so on. But the bottom line is, what it saying is you have mail.google.com, they have a certificate, which in turn was issued by GTS CA. Which is basically the Google Trust Store certificate authority.

Which in turn goes to the Google trust store root authority. So, there is one certificate, which has been created by the Google trust group, which has been given to the Google trust certificate authority, which in turn has been given to mail.google.com. It has a lot of information associated with it, it says when it will expire. And it also says the certificate is valid, because clearly the expiry date is after the present date.

It also talks about who has issued the certificate. It says it was issued by Google trust services. So, finally comes the question, who is the Google trust services? And why do I trust them? Because, I do not know anyone at Google. The point is, you are using a browser, you install the browser from somewhere. And inside the browser, it automatically has the information saying trust Google trust services. And in fact, your operating system itself, Windows, Mac, Linux, whatever it is, will have some kind of set of certificates that it is built to trust.

And you can put in a new trust certificate over there and say, trust all of the things that are signed by this root entity. Now, what that means is that when I download a browser, I am implicitly trusting all the entries that are there inside that browser. So, if for some reason, somehow, somebody managed to manipulate the browser and get a fake route entry into it, I am in trouble. Because, I would now start trusting things issued by that fake rooter. This is a very complicated problem to solve.

And in general, at some point, you just have to sort of buckle down and say, fine I trust that what I got over here was okay. Or at least the operating system that came installed with my laptop was done properly to start with. That itself had been hacked by someone and modified, there is not much I can do. And that is, literally the truth. But at this point, what it is saying is because your browser now trusts Google trust services, therefore everything else which is signed by Google trust services is also trusted by you. And you know that therefore this is actually mail.google.com.

(Refer Slide Time: 13:18)



Chain of Trust

- Chain of trust
 - mail.google.com issued certificate by
 - GTS CA1C3 issued certificate by
 - GTS Root R1
- GTS Root R1 certificate stored in Operating System or Browser
 - Do you trust your OS? Do you trust your browser?
- From there on a secure (crypto) chain

So, there is this concept of a chain of trust. mail.google.com has issued a certificate by GTS certificate authority, which in turn will issue a certificate by GTS root. And that would be stored either in the operating system or the browser. And like I said, do you trust your operating system? Do you trust your browser? There has to be some thing that you say, fine, this is the bare minimum, I will trust this and from there we will be able to (())(13:43).

(Refer Slide Time: 13:45)



Potential problems

- Old browsers
 - Not updated with new chains of trust
- Stolen certificates at root of trust
 - Certificate revocation, invalidation possible
 - Need to ensure OS, browser can update their trust stores
- DNS hijacking
 - Give false IPs for server as well as entries along chain of trust
 - But certificate in OS will fail against eventual root of trust

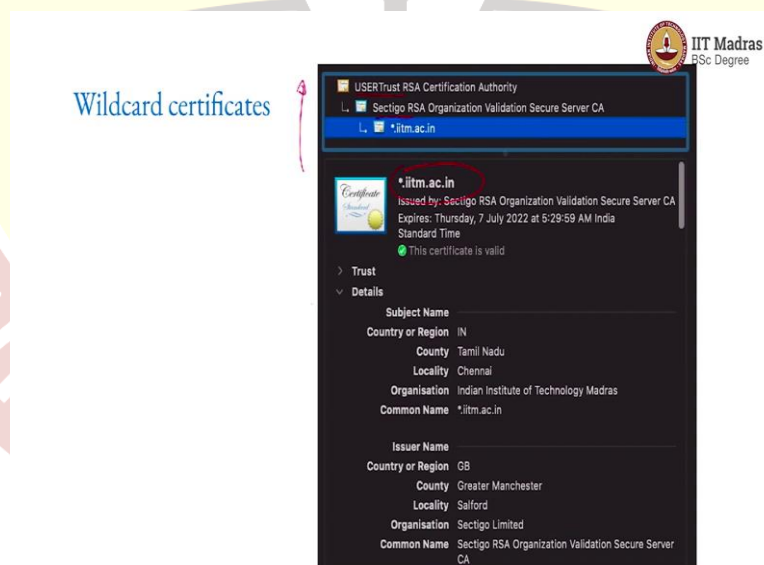
There are potential problems. So, for example, the GTS route might not be known to older browsers, because Google trust services itself was set up only a few years back. And older browsers, and when I say old, I am talking about really old, more than 10 15 years old. Might not even have been updated with new chains of trust.

Now, the worst case scenario is if someone steals a certificate at the root of trust. Which means that somebody setting up a root of trust actually has to have like, very, very serious security setup to ensure that such a thing cannot happen. If it is stolen, it can cause all kinds of problems, because now they can start issuing certificates that are essentially fake. And we will be trusted by all the browsers that trust this root of trust.

Now, what about the domain names? After mail.google.com I am going to that website based on some DNS lookup. If I end up having a broken DNS system, or somebody manages to modify my DNS settings, I might end up going to a different server. But the good thing is that the certificate that is already stored in my operating system or in my browser, will eventually fail. And it will say something is wrong. I am trying to connect to the server, it looks like the Gmail server, it says it is the Gmail server. That is what the DNS lookup says, then the certificate is wrong.

And it will pretty much, at that point logged out and say you cannot connect. This is usually rare, because these kinds of hacks are very difficult to do. But they are possible at least in theory.

(Refer Slide Time: 15:23)



Now, there is also a notion of something called a wildcard certificate. For example, iitm uses that. There is a certificate that has been issued for *.iitm.ac.in. So, anything www.iitm.ac.in, e.iitm.ac.in, academic.iitm.ac.in, all of them can use the same certificate. And in this case, it is issued by another root of trust. So it not GTS, it is not Google trust service. It is by something else, a company called Sectigo, which in turn has its authorization from another company called USERTrust.

So the entire chain is available, visible over here, for anyone who wants to verify it. It also gives the details of the subject name and also the issuer, issuer name of the certificate.

(Refer Slide Time: 16:09)



Impact of HTTPS

- Security against wiretapping
- Better in public WiFi networks

Negative:

- Affects caching of resources (proxies cannot see content)
- Performance impact due to run-time encryption



So, what is the impact of using HTTPS? It, of course, has positives, the main positive being that now the link between you and the server is secure against wiretapping. Meaning that even if you are on a public WiFi network, let us say in a coffee shop or somewhere else, as long as the server to which you are connecting uses HTTPS, you can be reasonably sure that anybody else who is listening in on your conversation will not be able to get important information related to this particular link.

So, at least on public WiFi networks, you need to make sure that you are actually connecting on HTTPS. Now, nothing comes for free. HTTPS can have a little bit of a negative impact, especially in terms of the performance. Meaning that you are performing runtime encryption. And the other negative impact is on caching. And what I mean by caching is a proxy server that sits between you and the, between the client and the server could have cached some of the common files.

Let us say, certain images or JavaScript files or other things. But when you are using an HTTPS connection, the information about your request does not get seen by any proxies, you cannot use a proxy. The proxy can just act as a pass through and send the information through to the server without being able to see what is inside the request. Because all that is encrypted. Which means that a proxy cannot respond to any requests that you have. So, caching also becomes a bit of a problem. It is a bit of a trade off as usual. And finally, the benefits from security are considered worthwhile.