CS 240: Secure Communication using HTTPS Transcript

[00:00:00] So now that we understand the idea of secure key exchange, let's look at how HTTP S uses secure key exchange to achieve secure encrypted communication over the web.

Start visual description. The professor demonstrates the concept of secure key exchange and how HTTPS uses it to achieve secure encrypted communication over the web. End visual description.

[00:00:15] So we're all familiar with uh http S URL S, so http S colon slash slash whatever.

[00:00:22] And so if, if you type a http S URL into a browser and hit, enter the process that occurs at that point is as follows.

[00:00:31] First of all, the web browser will establish a connection to the web server.

Start visual description. The professor shows the process that occurs when an HTTPS URL is typed into a browser, including the establishment of a connection to the web server and the initiation of the TLS handshake. End visual description.

[00:00:36] And because we're using HTTP S and want to have encrypted data going across the network.

[00:00:43] The first step in the communication between the client and the server is something called the HTTP S or the, the TLS handshake.

[00:00:51] And so the handshake process is a sequence of messages that the client and the server exchange where they're trying to set up the, the encrypted uh communication channel. So, they're trying to set up the encryption.

[00:01:04] And so um really, it's just a more detailed version of the secure key exchange algorithm that we just talked about.

[00:01:13] So on this slide, we can see in more detail how the TLS handshake works So in this case, we have the client and the server.

Start visual description. The professor explains the detailed steps of the TLS handshake, including the generation and exchange of random numbers between the client and the server. End visual description.

[00:01:20] And the first step in the, the TLS handshake is the client is going to generate a random number and send it to the server.

[00:01:29] Now, at this point, the communication is not encrypted.

[00:01:32] So this is this random number is exchanged in plain text.

[00:01:36] So the client sends the server a random number and we're going to call that random number, the client random because it was generated by the client.

[00:01:43] Now, when the server receives the client random number, the server is going to reciprocate with a random number of its own.

[00:01:49] So the server will generate a random number which we call server random and sends it back to the client.

[00:01:57] And again, this is still in plain text after the server sends its random number to the client.

[00:02:03] The next thing that the server will do is it will send its public key to the client.

[00:02:10] And so what we're trying to set up here is a secure key exchange.

Start visual description. The professor describes how the server sends its public key to the client and how the client responds with an encrypted pre-master secret. End visual description.

[00:02:13] So the server sends the client, it's public key.

[00:02:17] And then in response to that, the client then sends the server another random number which we call the premaster secret.

[00:02:25] But this time the random number, the premaster secret number is encrypted with the server's public key.

[00:02:32] So once the client has the server's public key, it can start to, we can start to use encryption.

[00:02:37] And so the random, the premaster secret is, is sent from the client to the server encrypted.

[00:02:42] And then the server, when it receives it decrypt, the premaster secret using its private key.

[00:02:48] So now both the client and the server have three random numbers in their possession.

[00:02:54] And so what each side then does is it takes the client random, the server random and the premaster secret and they use those random numbers as in input to an algorithm that would generate a symmetric key for an algorithm like AES.

[00:03:12] And so because the client and the server both can use the same algorithm to, to generate the symmetric key, they'll both have the, the symmetric key in their possession at that point.

[00:03:21] And then of course, once they both have the symmetric key, then then they can freely exchange encrypted data back and forth uh with each other.

Start visual description. The professor illustrates how the client and server use the exchanged random numbers and the pre-master secret to generate a symmetric key for encrypted communication. End visual description.

[00:03:30] And so every time uh a web browser downloads a file from a web server, this, this TLS handshake occurs and so this happens a lot.

[00:03:46] Now, the next question we might have is um OK.

[00:03:50] It's great that we can have encrypted communication between a client and the server.

[00:03:56] But another thing that you're interested in when you're interacting with a web server is you would really like to know that the web server, you're communicating with is who you think they are.

[00:04:06] Cuz it, it's um it's not that hard for somebody to impersonate an identity that that's not there.

[00:04:13] So somebody might create a fake website or something that, that claims to be your bank.

[00:04:17] And you know, if, if they're really good with uh their, their website design, then maybe you'll actually believe that it's real and, and uh share with them information you shouldn't.

[00:04:30] And so being able to authenticate or identify who the server is that you're talking to from the client perspective is, is really important.

[00:04:38] So the question is, how can a client verify the identity of the server? And the answer to that question is something called a public key certificate.

Start visual description. The professor discusses the importance of verifying the identity of the web server using a public key certificate and the information contained within the certificate file. End visual description.

[00:04:48] So when you create a website, typically, you also need to obtain a public key certificate and install that on your website so that clients can, can download your public key certificate and verify your identity.

[00:05:04] So when we talk about a public key certificate, we're really just talking about a file called a certificate file and inside the certificate file, you've got a couple of things.

[00:05:13] One is you have the public key of the, of the server. So here on the right, um the server has a public key and a private key.

[00:05:22] And so uh part of the handshake, as we saw the TLS handshake was the server is going to send its public key to the client. So, the public key for the server is actually inside the certificate file.

[00:05:37] So that's the first piece of information that's in the certificate file.

[00:05:40] The second piece of information is the identifying information for the server.

[00:05:44] So there's a, a bunch of information listed in the certificate uh about, you know who this this server is.

[00:05:51] So it would specify things like their organization name, any other um information that, that they wanted to include in their certificate.

[00:05:59] And it would also typically include their internet domain name, like google.com or Zions bank.com or, or whatever that might be.

[00:06:08] And so those two things together uh constitute what's called a certificate.

[00:06:16] So um when we talked about the TLS handshake in the previous slide, we there was a step number three where the server sent its public key over to the client.

[00:06:28] So what's actually happening there is the server is actually sending its um public key certificate to the client.

[00:06:35] And of course, inside the certificate there is the server's public key.

[00:06:39] So actually, it's, it's a little bit more than just the public key that's being sent there.

[00:06:43] It's, it's actually the whole certificate.

[00:06:47] And so during the, the TLS handshake, this whole certificate goes to the client and then the, the client can get the public key out of it so it can do the encryption uh the cure key exchange.

[00:06:57] But it also can look in there and see who, who is this server who owns the server that I'm talking to and validate, validate that it is who they, they think it is.

[00:07:11] Now the next question uh Once you understand that certificate exchange is how do we know that the server is telling the truth? So, the server is going to give the client a certificate file that, that, that states their identity.

[00:07:24] But how do we know that the server is not lying about their identity? How, how do we know that they're not claiming a false identity and or another way to put that? It would be. Why do I, why should I trust the, the certificate that the server sends to me? So, the answer to that is something called certificate authorities.

Start visual description. The professor explains the role of certificate authorities in validating the identity of the server and the process of obtaining a public key certificate. End visual description.

[00:07:42] So the idea with certificate authorities is that if, if I need to get a, a public key certificate for my website, for example, what I do is I go to a trusted organization,

a well-known trusted organization that um creates certificates and, and distributes those to people who want them.

[00:08:02] And so the notion is that certificate authorities are the people that create the certificates and um they're well known enough that people trust the certificates that are created by these organizations.

[00:08:14] So we're talking about companies like uh Go Daddy or Digic Cert or, you know, there's, there's a number of companies that, that you can buy AAA digital certificate from.

[00:08:24] And there's also an organization called Let's Encrypt that will actually give you um they'll, they'll create free certificates. So, you don't have to pay, let's encrypt any money.

[00:08:36] So if you don't want to pay these other companies uh, because they, they normally do charge to, to create these certificates because that's a part of their business.

[00:08:43] But let's encrypt will actually create, uh, certificates for free.

[00:08:47] The only catch is that, uh, if you get a certificate from, let's encrypt, um, they expire every th, every three months.

[00:08:54] And so you have to renew your, your certificates every three months, but it's not that hard to renew a certificate through them.

[00:09:00] So that, that's a good way to, to get a certificate.

[00:09:04] So the idea is because the, the certificate files come from these, these larger trusted organizations then that gives the client trust that yeah, I can, I can trust the identity that's inside this certificate file.

[00:09:21] Ok. So, we said that um, if, if somebody who has a website needs a certificate, they need to go to a certificate, authority to get it.

[00:09:30] So the process for doing that is that the person who owns the server would need to follow a few steps.

[00:09:37] First of all, they would need to generate a public private key pair.

[00:09:41] And so you would use a tool like OpenSSL or something to, to create your own public private key pair.

[00:09:46] If you don't already have it, if you already have it, you can use the one you've already got.

[00:09:50] And once you've got your public private key pair, you'll of course, tuck away your, your private key in a file somewhere that's secure and you're not going to share that with anybody, not even the certificate authority.

[00:10:02] Um And then what you'll do is you'll send your public key and also the identifying information that you want to put in your certificate file, and you'll send uh your public key and your identifying information to, to a certificate authority.

[00:10:18] And when, when you do that, they call that AAA certificate signing request.

[00:10:22] So you're, you're asking them to, to create a certificate for you and to, to digitally sign it.

[00:10:28] And it's through the, the digital signing process that, that people can trust that this certificate file actually came from a trusted certificate.

[00:10:36] Authorities is because the certificate file actually has the, the, the certificate authority, uh digital signature on it.

[00:10:44] So once you send a CS R or a certificate signing request to a certificate authority, they'll go ahead, and um validate your identity. So, they'll go through some process to uh convince themselves that you are who you say you are.

[00:10:58] And there's a variety of mechanisms they can use to do that.

[00:11:00] But once they're convinced that you're telling the truth about your identity, then they'll go ahead and create a certificate file with your public key and your identifying information in it.

[00:11:09] And then they'll, they'll digitally sign it and then send it back to you.

[00:11:12] And then you would install that certificate file on your website.

[00:11:15] And then any time a client or a browser comes to your website, they would receive a copy of your certificate and that's where that whole handshake process comes in.

[00:11:25] OK. So, the next question becomes, I just said that the certificate, authority is going to digitally sign the certificate file so that it can be trusted by clients.

[00:11:40] So the next thing I want to do is talk about how does that digital signature process work? What does it mean to sign a file with a digital signature? And why should the client trust that? And so, um in the next video, we'll talk about um how the digital signature process works.

[00:12:01] So you can get a, a certificate from a certificate authority as we just described.

[00:12:05] Another thing you can do if you want to is create what's called a self, a self-signed certificate means you can make your own certificate so you can act as your own certificate authority if you want to think of it that way.

[00:12:16] Now you wouldn't use a self-signed certificate in a production environment on a real web server, for example.

[00:12:23] But what you might need is certificates that you can use as a software developer as you develop and test your, your software.

[00:12:31] So you could either, you know, create your own certificates or you could go to let's encrypt and uh get free certificates from them or you, you know, you can get your certificates however you want.

[00:12:40] But one approach is to create your own.

[00:12:42] And if you do create your own self signed certificates by default, web browsers don't trust a self-signed certificate.

[00:12:50] And so if you do use self-signed certificates, you'll have to configure your web browser to trust your self-signed certificate.

[00:12:56] And there's ways you can do that.

[00:12:58] Uh It's a different process depending on the browser.

[00:13:00] But assuming you've created a self-signed certificate, install it on your, um your web server, um and then configured your browser to trust it.

[00:13:09] Then you can, can do development and testing using a self-signed certificate.

[00:13:14] There's lots of tools out there for creating self-signed certificates.

[00:13:17] Uh There's a really popular one called open SSL.

[00:13:21] So open SSL is a really popular library that implements the TLS or SSL protocol, which is at the heart of HTTP S.

[00:13:32] And it also has a set of tools for doing lots of things such as managing certificates and so on this slide.

[00:13:39] For example, you can see the open SSL command that you would need to run to generate your own uh certificate.

[00:13:46] Now, we don't really need to go through the, the details too much of that, but just, just realize that when we create the, the self-signed certificate, we have to give them our public key, we have to give them our identifying information.

[00:13:58] And in this case, we have to tell them the expiration date.

[00:14:01] So how long do we want the certificate to last before it expires? In this case, it's a 10-year certificate which is, is egregiously long. But for development testing, it doesn't matter.

[00:14:11] So just, just so you know, that you can create uh a self-signed certificate if, if you need to.