Transport Layer Security

Hi. In this video, we're going to be looking at TLS, Transport Layer Security. This is a protocol which allows a client to communicate with a server, agree a symmetric key for encryption, and to also have confidence that the server is authentic. TLS is a term which you might have heard already.

It's commonly used on the internet to deliver secure pages. For example, HTTPS relies on TLS. It should be noted that TLS builds on a previous protocol called SSL. The community have somewhat held on to the terminology SSL. So if you do see SSL, you can think of this as being similar.

In saying that, TLS is much more secure than SSL. And it's certainly inadvisable to actually use SSL protocol. In particular, TLS 1.3 is the most recent version at the time of recording. This was finalised in 2018 and allows us, in a fairly quick manner, to be able to agree symmetric key to be used for encryption and, also, for us to authenticate the server.

So now, we'll turn our attention to the detail as to how the protocol itself functions. Before we do that, just note that we are doing this at somewhat of a high level. We don't necessarily have the time to go into this in great depth. However, if you are interested, I can put some resources below where you can follow up for more detail. We'll now break down how TLS 1.3 works.

The most important part of TLS that we're going to spend some time on is what's referred to as the handshake. TLS, effectively, allows a client and a server to agree a shared key, a symmetric key, which can be used for encryption and, also, allows us to have some confidence that the server is who it claims to be. So there is an authentication element to it as well.

This is achieved through a handshake process. A handshake basically refers to a series of messages back and forth, which allow us to achieve these two objectives, that is to agree a symmetric key for encryption and, also, to have some confidence that the server is who they are claiming to be--

that authentication element.

So the first message that's sent is the client hello. Within this, we've got some information. We have the supported cipher suites of the client. This is, effectively, the set of primitives and protocols within cryptography that the client is able to support. This can include key exchange algorithms, such as variations on Diffie-Hellman. It can include digital signature algorithms, such as RSA-PSS.

Recall that PSS is a particular form of padding, which adds masking and improves the security of using RSA by itself, which is generally inadvisable and a variety of different hash functions and so forth. The other thing that it sends along is a whole bunch of possible key shares. These are, effectively, values, which can be used to agree a symmetric key over insecure channel.

Effectively, it depends on which form of Diffie-Hellman or other key agreement protocol they're using. But they would provide a list, which would fit with those protocols. This is sent over to the server. The server can then send back the server hello.

And again, this includes a few bits of information. It says, OK, this is the best cipher suite that I think I can meet you in terms of what you can provide and, also, provides a key share. So if we think of this like a Diffie-Hellman protocol that's being used, then these key share values are going to be the values that we share over an insecure network that allow us to agree on that key without having to share the key itself directly or anything, which allows an interceptor to directly establish that key.

It also sends over its digital certificate. And it sends over a finished message. At this point, unlike with TLS 1.2, these are both encrypted because, if we look at this stage, we have sufficient information to be able to determine what the symmetric key is going to be. Because the client has shared that in advance, it's trying to take a bit of a leap guessing what the server might be able to support.

And that's one of the things that makes 1.3 faster than 1.2, which would normally take more communication back and forth before it was able to arrive at that information. Of course, if there is no crossover in terms of what the client support versus what the server supports, then there are a number of different routes. We can either just close that communication off and say it's not possible. Or it may go back and see perhaps can you do TLS 1.2 and move down from there. We're not too interested in the detail of that.

But obviously, there is an immense amount of detail on this protocol out there, not least of all the actual specification and the request for comments document if you're so interested. So at this point, it can send that information back to the client. And the client can then start to verify

things.

So it can generate what the shared key is going to be because it now knows what the key exchange algorithm and the key share is. And it can also take that encryption then off of the digital certificate and the finished communication. And it can verify that digital certificate. So if it is happy with all of those checks, then the client can send over a message to say, OK, that's me.

I'm finished. And I want to request certain information, for example, an HTTPS request. And the server can then send back the answer. And at this point, recall this is all now corrected using the symmetric key, which was agreed using the key exchange algorithm at the start. That's a very brief overview of how TLS 1.3 works.

Having now covered the steps involved in TLS 1.3, let us now turn our attention to a couple of properties that I'd like to highlight. One is the property of forward secrecy. With previous versions of TLS or with SSL, if a secret key was compromised, then past communications could also be compromised. This is obviously less than ideal.

In contrast, forward secrecy means that, if at some point in the future one of your keys is broken, it's not going to be able to go back very far to decrypt any communications which have taken place before. The reason for this is the use of the Diffie-Hellman structure and the fact that the keys that are being generated there are ephemeral. This means that they're only used within the context of that communication. This is in contrast to TLS 1.2 where, for example, RSA was permitted as part of our key sharing process.

Another property to point out is the increased security in terms of the confidentiality and the resilience against attacks of the cipher suites. In TLS 1.2, there are a wider range of cryptographic protocols and parameters that which are usable. However, some of these were being used in a less secure manner. As a result, these have been removed to some extent within TLS 1.3. As a result, this provides further layers of security.

That's it for TLS. I hope you've enjoyed the video. And I'll see you next time.