**Network Security**

**Professor Gaurav S. Kasbekar**
**Department of Electrical Engineering**
**Indian Institute of Technology, Bombay**

**Week - 05**

**Lecture - 30**

**Secure Sockets Layer (SSL) and Transport Layer Security (TLS): Part 1**

Hello, recall that TCP, or Transmission Control Protocol, is a popular transport layer protocol used on the internet. It provides various mechanisms, such as reliability, congestion control, and flow control. So, it performs all these functions: reliability, flow control, and congestion control. But TCP does not have any security mechanisms. So, in this lecture and the next lecture, we will discuss a protocol, which adds security mechanisms to TCP.

That protocol is known as Secure Sockets Layer (SSL). It has been standardized, and the standardized version is known as Transport Layer Security (TLS). So, it's the same protocol. So, there are two different versions. One is the original version, which is SSL, and its standardized version is known as TLS.

SSL adds the following security services to TCP. One security service is confidentiality. It encrypts data before sending it. So that results in confidentiality. Then it also provides message integrity.

In particular, for each packet, there is a Message Authentication Code (MAC) added, which provides message integrity. Also, it has mechanisms which ensure that an intruder on the path between a sender Alice and a receiver Bob is not able to reorder packets or delete packets and so on. So, it provides mechanisms for these. It also adds server authentication, which means that the server of a TCP connection can prove to the client that the server is indeed legitimate. Similarly, SSL adds client authentication, which means the client is able to prove to the server that it is indeed the legitimate client. So, we'll discuss how these different security mechanisms are implemented.

So, TCP does not have any security mechanisms, but SSL adds all these security services to TCP. A slightly modified version of SSL, called Transport Layer Security (TLS), has

been standardized. So, it's the same protocol. SSL is the original protocol, and its standardized version is TLS. We'll use the terms SSL and TLS interchangeably.

So, SSL or TLS is extensively used on the internet to secure web traffic or HTTP traffic. For example, by websites like Google, Amazon, eBay, and so on. Recall that HTTP is a protocol that runs on top of TCP. So, to secure HTTP traffic, we use SSL; the security mechanisms provided by SSL. So, SSL is especially useful when some sensitive information needs to be transferred.

For example, in e-commerce transactions, credit card details are often provided. The server sends a form in which the credit card details have to be entered. So, these credit card details should not be leaked to some intruder who is eavesdropping on the channel between the client and the server. So, SSL can be used to securely transfer credit card details. Another example is passwords.

Users enter passwords when logging into websites, such as Gmail, Yahoo Mail, and so on. So, for securely transferring these passwords, we use SSL. When SSL is used by the browser, the URL begins with https: instead of http:. So, that's an indication that it's a secure connection, secured using SSL. So, here's an example that motivates the need for adding security mechanisms to TCP.

Suppose a client, Bob, visits the website "Alice Incorporated" and wants to order some perfume. So, the way that the exchange happens is as follows. The website sends a form to Bob in which he enters the type of perfume, and the quantity required and Bob's address to which the perfume has to be dispatched, and Bob's credit card information for billing purposes, and so on and so forth, and other relevant information and Bob submits the form. So, in this exchange, suppose no security mechanisms are used. In that case, what are some examples of attacks by an intruder, Trudy, in this scenario, assuming that we don't use any security mechanisms?

So, if Trudy eavesdrops on the channel between Alice and Bob, in that case, Trudy can sniff Bob's messages using a packet sniffer, extract the credit card information entered by Bob from the sniffed messages, and use this credit card information to make purchases. This is possible because no confidentiality is used. So, this illustrates why the information that is exchanged between Alice and Bob needs to be encrypted. If there is no confidentiality used, then, for example, credit card information will be stolen and may be used to make purchases. Another example of an attack is that Trudy may modify Bob's messages. This modification is possible because there is no message integrity used.

One example of a modification is that Trudy modifies the message of Bob and has him order 10 times more perfume than required. So, this will lead to Bob being billed by a huge amount and 10 times more perfume than required being sent to him. So, someone who wants to cause mischief can modify messages in this way. And this modification is possible because there is no message integrity used. So, this motivates the need for adding message integrity to TCP.

Another example of an attack by Trudy is the following. Trudy may intercept all messages that are being exchanged between Alice and Bob. For example, suppose Trudy controls a compromised intermediate router on the path between Alice and Bob, and Trudy then sends a fake webpage with Alice Incorporated's name and erroneous details. So, Bob thinks that he's downloading the webpage of Alice, but actually, it is a fake webpage created by Trudy with just the name being "Alice Incorporated," and the other details being erroneous. So, Trudy can modify the details on Alice's website to some fraudulent details, and this attack is possible because no server authentication is used.

The server does not prove to the client that it is legitimate. So, this motivates the need for adding server authentication to TCP. Such attacks are prevented by SSL because it adds confidentiality, message integrity, and server authentication to TCP. These are just some examples of possible attacks. So, there are many other attacks that can be launched if no security mechanisms are used.

So, all these attacks are prevented by SSL. So, we have been discussing so far SSL for securing web connections, but note that SSL secures TCP, which is the transport layer protocol. So, apart from HTTP, SSL can also be used to secure any application that uses TCP. One example is file transfer, which also happens over TCP. So, SSL can secure file transfer, and the secured version of file transfer is known as FTPS, which stands for file transfer over SSL.

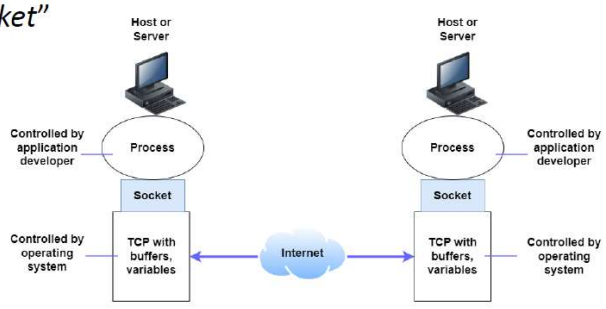
So, FTP is the original protocol without any security mechanisms, but if file transfer is done after securing it using SSL, then the protocol is FTPS, a secure file transfer protocol. So, we now discuss where SSL appears in the protocol stack. So, we discussed the internet protocol stack consisting of five layers: application, transport, network, link, and physical. So, in this protocol stack, where does SSL reside? So, for that, we discuss the concept of a TCP socket.

Consider an application that uses TCP without SSL. So, this shows the protocol stacks at the client and at the server. So, this is the client and this is the server, and this is the

application layer at the client, and this is the transport layer at the client. So, the application process runs in the application layer, and TCP runs in the transport layer. So, this transport layer and the lower layers are controlled by the operating system, whereas the application process is controlled by the application developer. Similarly, on the server side, we see that there is an application process that is controlled by the application developer, and there is the transport layer and lower layers, which are controlled by the operating system.

So, in particular, TCP with its buffers and variables that resides in the transport layer on each side. Now, in this context, the application process in the application layer at each end sends messages into and receives messages from the network through a software interface called a socket. The socket is illustrated here. It is an interface between the application process and the TCP process. So, this socket is illustrated in this picture on each side, the client and the server.

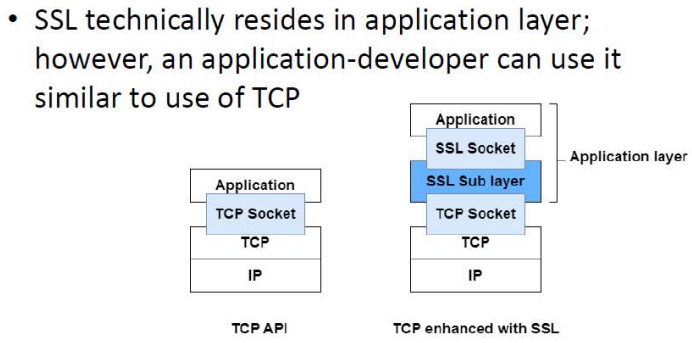
- Consider an application that uses TCP (without SSL)
- Application process (in application layer) at each end sends messages into and receives messages from network through a software interface called *"socket"*




So, the socket is an interface between the application process and the TCP process. So, this socket provides an API: Application Programming Interface. It provides commands such as, for example, send a message, receive a message, open a connection, close a connection, and so on. So, the socket is an API, which provides all these commands to the application process. And these commands are implemented by TCP, which resides below the API, below the socket API.

So, that's the context in which we want to add SSL. SSL provides an API similar to TCP's API. It provides commands to establish and close connections, send messages, receive messages, and so on. All these commands are included in the API of SSL. The figure on the left shows the TCP socket or API, and the figure on the right shows the protocol stack with SSL added.

So, on the left side, the application process directly talks to TCP via the TCP socket. But on the right side, we see a scenario where SSL is used. In this case, there is a TCP socket, which is the same as in the figure on the left. There is an SSL sub-layer over it. This SSL sub-layer has different security mechanisms for providing confidentiality, message integrity, authentication, and so on.




So, this SSL sub-layer resides above the TCP socket. The SSL sub-layer uses the commands provided in the TCP socket and adds its own security mechanisms to provide secure commands to the application process. So, it provides an SSL socket through which the application can access the services provided by SSL. So, now with this SSL socket, the application process can again perform functions similar to what it can with a TCP socket. For example, open a connection, close a connection, send a message, receive a message, and so on.

But now these commands are performed securely. If the application says to send a message, then that message will be encrypted, and message integrity will be added to it, for example. So, this SSL socket provides commands similar to the commands that TCP provides, but with security built into those commands. So, in the protocol stack, SSL technically resides in the application layer, as shown in this picture here. So, the SSL sub-layer resides in the application layer, but the application developer can use it similarly to the use of TCP.

So, the only difference is that instead of using the commands provided by TCP, the application has to use the commands provided by SSL. But these commands are similar to those provided by TCP. So, from the application developer's perspective, this scenario is not very different from directly using TCP. It is similar; the commands are similar to those in TCP, but security is provided in them. So, it's better to use SSL since security is available.

First, we study a simplified version of SSL to gain understanding, and then later we study the actual SSL. So, SSL, as well as its simplified version, has three phases. One is the handshake phase, then the key derivation phase, and data transfer. So, in the handshake phase, which algorithms will be used by SSL, these are negotiated in the handshake phase. And then, some material is exchanged, using which keys will be derived in the second phase.

So, in the handshake, some information is exchanged, using which keys can be derived. Then, in the key derivation phase, keys are derived by the client and server, which can be used for providing encryption, message integrity, etc., in the data transfer phase. Then, finally, we have the data transfer phase, in which the actual data that the client and server want to exchange is transferred securely using the secret keys that were derived in the key derivation phase, and once the data transfer is complete, the connection is closed. So, we'll first discuss the simplified version. For example, there is no negotiation of the algorithms that will be used.

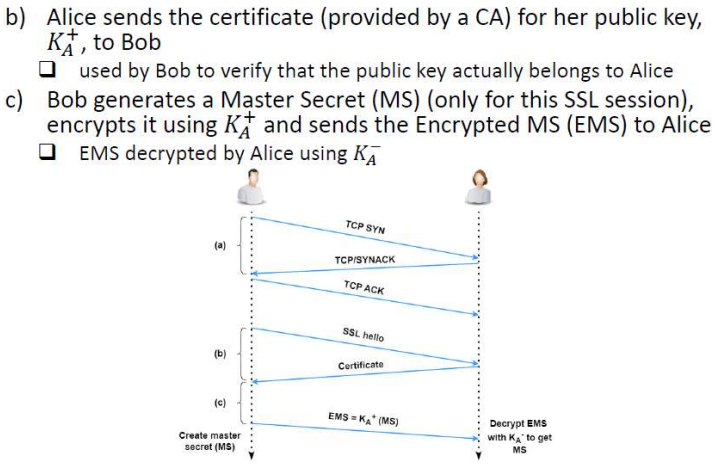
So, we'll discuss that negotiation later when we discuss the actual SSL. So, we start with the simplified version. Consider a communication session between a client, Bob, and a server, Alice. Assume that the server has a certificate for its public key from a certification authority. For example, the server might be Google's server, and the client may be connecting to Google's server.

The client may be a user who connects to Google's server. And Google has a certificate for its public key from a certification authority. So, we now discuss this communication session, how it is executed using SSL. So, in the simplified version of SSL, the first phase, that is the handshake, is performed as follows. This shows the handshake.

So first, in step (a), Bob and Alice establish an ordinary TCP connection. So, for establishing a TCP connection, three messages need to be exchanged. The first message is sent by the client to the server, which is the 'TCP SYN' message. SYN stands for synchronization. Then the server responds with the 'TCP SYN ACK' message.

And then the client sends 'TCP ACK'. That's a response to the 'TCP SYN ACK' message sent by the server. So, using these three messages, a TCP connection is initiated. So, this initiation is just like the initiation of an ordinary TCP connection. There are no security mechanisms here.

So, each side uses a sequence number, and this SYN message is used to share the initial sequence number by the client. In this second message, the server shares the initial sequence number used by the server. So, this is just an ordinary TCP connection initiation without any security mechanisms. So, in the second phase, the client, Bob, initiates an SSL connection by sending this message, 'SSL hello'. And then the server, Alice, sends the certificate provided by a CA to Alice for a public key. Let's denote the public key by $K_A^+$.




So, Alice sends the certificate to Bob, and Bob uses this certificate to verify that the public key actually belongs to Alice. So, recall that the certificate has Alice's name, address, her public key, and a digital signature by the certification authority. So, Bob verifies that the public key actually belongs to Alice using the certificate. This exchange happens in phase (b), where the certificate is shared by Alice to Bob. Then, in step (c), Bob generates a master secret (MS) used only for this SSL session.

So, this master secret is different for different SSL sessions. And Bob encrypts the master secret using Alice's public key $K_A^+$, and the encrypted master secret (EMS) is sent to Alice. So, this shows the transfer of the encrypted master secret. So, the master secret is encrypted using Alice's public key and transferred from the client Bob to the server Alice. And then, the encrypted master secret is decrypted by Alice using her private key, which is $K_A^-$.

So, at this point, Alice decrypts the encrypted master secret with her private key to get the master secret. So, after this message has been transferred and after Alice has decrypted it, Alice and Bob now have a secret shared between them, which is the MS, master secret. This is used to create the keys that will be used for achieving encryption and message integrity. This is the handshake phase, where the certificate is transferred from the server

to the client, and then a master secret is exchanged between the two sites. In particular, the client transfers the master secret to the server after encryption.

So, at this point, the material that is required to generate keys has been shared. So, this material is the master secret that will be used to derive keys in the second phase. So, now we discuss the second phase of SSL, which is the key derivation phase. So, the master secret that was transferred from the client to the server in the first phase, that master secret is used by Alice and Bob to generate four symmetric keys. These keys are the following: $E_B$, $E_A$, $M_B$, and $M_A$.

- $E_B$: key used for encryption of data sent from Bob to Alice
- $E_A$: key used for encryption of data sent from Alice to Bob
- $M_B$: key used for message integrity of data sent from Bob to Alice (used as the authentication key to generate MAC)
- $M_A$: key used for message integrity of data sent from Alice to Bob

$E_B$ is the key used for encryption of data sent from Bob to Alice. $E_A$ is the key used for encryption of data sent from Alice to Bob. $M_B$ is the key used for message integrity of data sent from Bob to Alice. So, $M_B$ is used as the authentication key to generate the message authentication code. So, recall that if a message is m, then the message authentication code is this, H(m,s). So, H(m,s) is the message authentication code for the message m, where s is a secret known only to the sender and the receiver.

So, this s is known as the authentication key. And in this context, $M_B$ is this authentication key used for message integrity of data sent from Bob to Alice. And similarly, $M_A$ is the authentication key used for message integrity of data sent from Alice to Bob. So, these four keys are generated using the master secret. Now, the question is why do we generate four different keys from the master secret instead of using the master secret itself as the key for all four tasks?

That is encryption on the two sides and a message indicator on the two sides. So, the reasons are as follows. In a given direction, if the same key is used for encryption of a block and for generating the message authentication code for that block, then the attacker may be able to use this fact and the knowledge of the two algorithms, namely the encryption algorithm and the message integrity algorithm, to get some information about the plaintext data. So, hence, it is considered good security practice to use different keys for encryption and message integrity. So, this is one reason for using different keys for encryption and message integrity in a particular direction.

Now, a question is: can we use the same encryption key and message integrity key for the two directions? So, if the same keys are used in both directions, then the attacker may be able to take an encrypted block from the client and send it back to the client as if it were sent by the server or vice versa. So, for this reason, different keys are used in the two directions. So, we discussed this earlier as well. In a particular direction, say from Bob to Alice, different sequence numbers are used for different packets that are sent.

But then an attacker may be able to take a packet with a particular sequence number, say N, that is being sent from Bob to Alice, and then play it back to Bob, pretending that it is the packet with sequence number N sent by Alice to Bob. So, such an attack is possible if the same keys are used in the two directions. So, for this reason, different keys are used for the two directions. So, for encryption of data sent from Bob to Alice, the key $E_B$ is used, whereas for encryption of data sent from Alice to Bob, the key $E_A$ is used. And similarly, the authentication keys used to generate MACs are different in the two directions.

$M_B$ is the key used by Bob to Alice, and $M_A$ is used by Alice to Bob. So, this is the key derivation phase. Then, the third phase is the data transfer phase. In this data transfer phase, Alice and Bob start sending the data with encryption and MAC generation done using the above keys. So, Alice and Bob start sending data to each other using encryption and MAC generation, which is done using the keys that were derived in the previous phase.

So, the MAC should be calculated over which part of the data? So, one way is to generate a message authentication code for the entire data and send it at the end of the transfer. So, for example, if Alice sends a webpage to Bob, then the message authentication code is calculated over the entire webpage and sent at the end. So, this method has the shortcoming that there is a large delay until the data can be used by applications. So, the sender has to wait until the entire data is transferred, only then the MAC can be sent.

And then the receiver has to wait until all the data has been received, and only at that point can the MAC be verified. So, this has this shortcoming. So, can we break the data into smaller parts and generate a MAC for each part? So, a better approach is this: the data stream in each direction is broken into chunks called records, and a MAC is appended to each record. So, for example, if a large website is to be transferred from Alice to Bob, then the website's data is broken into chunks called records, and a message authentication code is appended to each record.

To generate the MAC, Bob finds the hash of the record data and the authentication key, which is $M_B$. So, this is the usual scheme. The MAC for a message m is H(m,s), where this

$M_B$ is the secret key s, and the record data is m. So, the MAC is computed in the usual way. It is calculated as the hash of (data, $M_B$). That is the authentication key.

- To generate MAC, Bob finds hash of (record data, $M_B$)
- Also, Bob encrypts (record data, MAC) using $E_B$ and passes it to TCP for transport to Alice

Also, Bob encrypts (record data, MAC) using the encryption key $E_B$ and passes it to TCP for transport to Alice. So, in this way, message integrity is added to the record, and it is also encrypted. Now, can the message integrity of this scheme be breached by Trudy, who is an intruder controlling a compromised intermediate router? So, note that each record is individually protected using a MAC, but the intruder might still be able to reorder records or delete some records and so on. So, the message integrity of this scheme can be breached by Trudy by deleting records, replaying records, or reordering records.

For example, Trudy may capture two packets that are sent by Bob, reverse their order, and adjust the TCP sequence numbers, which are not encrypted, and send them to Alice. So, TCP has its sequence numbers, which are not encrypted. So, it's easy for an intruder to capture two packets, reverse their order, adjust the sequence numbers, and send them to Alice. Because of this attack, the order of packets will be wrong at the receiver. So, what is the scheme to defend against such attacks?

We discussed a scheme that can defend against such attacks in an earlier lecture. So, one scheme that can be used is to assign an SSL sequence number to each record. We assign a sequence number to each record, and a different sequence number is assigned to different records. An optimization that is performed is that the sequence number is not included in the SSL record itself, but instead the following scheme is used. Bob maintains a sequence number counter which begins at zero and increments for each SSL record that he sends.

- Sequence number included in calculation of MAC for a record:
  - ❑ MAC is hash of (record data, $M_B$, *current sequence number*)

Similarly, Alice maintains a counter that increments each time an SSL record is received. So, Bob and Alice know the sequence numbers because they have counters which keep track of the sequence numbers. The sequence number is included in the calculation of the message authentication code for a record. The MAC is the hash of (record data, $M_B$, the current sequence number). So, the sequence number is included in the MAC calculations so that no one can, for example, reorder packets or delete packets.

So, we will come to know of such attacks by the sequence number which is used for the computation of the MAC. Alice checks whether the MAC is correct using her own sequence number counter. So, this scheme is used so that we can defend against attacks, such as reordering, replaying, or deletion of packets. Now, what is an advantage of this scheme over that in which the sequence number is included in the SSL record? So, note that in this scheme used in SSL, the sequence number is not included in the packet.

But instead, the sender and receiver keep track of the sequence number using counters. So, an advantage of this scheme over the scheme in which sequence numbers are included in the SSL record is the following. That sequence numbers are included by TCP. We know that TCP has its own sequence numbers. So, these sequence numbers can be used to put packets in the correct order, and if there are any discrepancies, then they can be checked using the MAC.

The sequence number is included in the MAC calculation, so any discrepancies can be checked using the message authentication code. Hence, because of the presence of sequence numbers included by TCP, it is not necessary to include sequence numbers in the SSL record. So, we save some bandwidth by omitting the sequence number. So, this is one reason for not including the sequence number. Another reason is that the intruder can maintain a counter, just like Alice and Bob maintain counters.

So, the intruder can also maintain a counter that increments each time an SSL record is sent. So, the intruder knows which sequence number to expect. So, if the sequence number is included in the SSL record, then the intruder gets access to the ciphertext for a known plaintext. So, in general, a cipher is weakened if an intruder has access to some known plaintext and the corresponding ciphertext. So, in this case, the intruder can easily predict the sequence number, so it is not included in the record.

So, if it were included, then the intruder would gain access to some known plaintext, which is the sequence number that the intruder has kept track of, and the corresponding ciphertext, which is included in the packet. So, to avoid this attack, the sequence number is not included in the SSL record. Now, we will discuss the SSL record format. This format is shown in this figure. The type field in the record indicates whether the record is a handshake message, a message that contains application data, or a message used to close the connection.

Then, the length field is used by SSL at the receiving end to extract the SSL record from the TCP byte stream. Note that one SSL record need not correspond to one TCP packet.

So, TCP sends a stream of bytes, which is broken into different packets, and one SSL record may not correspond to one TCP packet. One SSL record may be spread over multiple TCP packets, or there may be multiple SSL records in a single TCP packet, for example. So, the version is the version of SSL, and these fields—type, version, and length—are not encrypted, so we can see that encryption is only done over the data and MAC, and the sequence number, which is not included in the SSL record.

So, encryption is done over the data, sequence number, and MAC, but the type, version, and length fields are not encrypted; however, any modification in them can be detected using the MAC, which is computed over these fields as well. So, modification in these fields by an intruder can be detected using the MAC, which is computed over the data, sequence number, secret key, type, version, and length. So, in summary, we started our discussion of the protocol SSL or TLS. We discussed the different phases in SSL, including the handshake phase, key derivation phase, and data transfer phase. We started with the discussion of the simplified SSL, and later on, we will discuss the actual SSL that is used.

So, in the next lecture, we'll discuss the actual SSL protocol. Thank you.