Network Security

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

Week - 05

Lecture - 31

Secure Sockets Layer (SSL) and Transport Layer Security (TLS): Part 2

Hello. Recall that we started our discussion of SSL and TLS in the previous lecture. We now continue that discussion. In the previous lecture, we discussed the simplified version of SSL. The actual version of SSL is slightly more complicated than the simplified version we discussed.

So, it includes, for example, negotiation of the algorithms that will be used for encryption and message integrity, and so on. So, we'll discuss the actual SSL protocol now. So, SSL is quite flexible. It does not mandate that a particular symmetric key encryption algorithm, public key encryption algorithm, or MAC computation algorithm be used. So, we can use one out of multiple possible symmetric key encryption algorithms, for example, DES, AES, and so on.

We can use different public key encryption algorithms, and we can use different MAC computation algorithms. So, which particular algorithm is to be used in a particular exchange? So, this can be agreed upon by Alice and Bob during the handshake. So, the sender and receiver can agree on which symmetric key encryption algorithm, which public key encryption algorithm, and which MAC computation algorithm should be used during the session. So, these can be agreed upon during the handshake.

What is the advantage of this flexibility over a scheme that mandates a particular symmetric key encryption algorithm, public key encryption algorithm, and MAC computation algorithm? So, the advantage of this flexibility is that if a specific algorithm is broken or a weakness is found in it, then another one can be used. For the same reason, several security systems offer similar flexibility. So, any one algorithm out of multiple choices can be used for different functions. It happens all the time that a particular algorithm is standardized; a particular security algorithm is standardized, but then later on, some weaknesses are found in it.

So, it is no longer considered secure. So, in such cases, that insecure algorithm can be replaced with another algorithm that is secure. So, for this reason, SSL provides this flexibility. The sender and receiver can use any algorithm out of a choice of multiple algorithms. Also, during the SSL handshake, that is the first phase, Alice and Bob send nonces to each other.

- Also, during SSL handshake, Alice and Bob send nonces to each other; these are used in computation of session keys E_B, E_A, M_B and M_A

These are used in the computation of the session keys, that is, E_B, E_A, M_B, and M_A. E_B, E_A, M_B, and MA have the same meanings as in the simplified SSL protocol. That is, E_B and E_A are the encryption keys, and M_B and MA are the authentication keys. So, these nonces are used to defend against connection replay attacks, which we will discuss in detail later on. So, we now discuss the actual SSL handshake.

So, it has 6 steps; these steps are as follows: First, the client sends a list of cryptographic algorithms that it supports, along with the client nonce, to the server. For example, the client may support AES, DES, and 3DES as the symmetric key encryption algorithms. Then, these three algorithms are included in this list of cryptographic algorithms that the client supports. The client also sends a client nonce to the server.

This is used to defend against replay attacks where an intruder records all the messages that are exchanged between the client and the server and later on intruder replays these messages. Later on, the intruder pretends to be the client or the server using these recorded messages. So, to defend against these attacks, we use a nonce on each side. So, in particular, the client sends a nonce called a client nonce. And in step 2, the server selects one symmetric key algorithm, one public key algorithm, and one map computation algorithm, and possibly other algorithms from this list that was shared by the client.

And the server sends its choices to the client. And the server also sends a certificate and a server nonce to the client. So, these two nonces, client nonce and server nonce, are used to defend against replay attacks, as we discussed. And the server sends a certificate, and this certificate is used to extract the public key of the server. Now, in step 3, the client verifies the certificate; that is, recall that the certificate has the server's name, address, and public key, and there is a digital signature by the certification authority.

So, using this digital signature, the client verifies that the certificate is indeed that of the server with whom the client wants to communicate. And if this verification is successful, then the client extracts the server's public key from the certificate. Client generates a random pre-master secret, or PMS. So, this pre-master secret will be used to generate the keys E_B, E_A, M_B, and M_A. So, this is the material that will be used to derive the secret keys.

So, this random pre-master secret is encrypted with the server's public key and sent by the client to the server. So, the client encrypts the PMS with the server's public key and sends the encrypted PMS to the server. So, this is step 3. Then, in step 4, the client and server independently compute the master secret from the PMS and the nonces. This master secret is used to generate the four keys, that is, the two keys for encryption in the two directions and the two authentication keys in the two directions.

So, these four keys, E_B, E_A, M_B, and M_A, are generated from the master secret. So, notice a difference related to the simplified SSL that we discussed. So, in the simplified SSL, the client directly sends the master secret encrypted using the server's public key, and then the four keys are derived from the master secret. So, in contrast here, the client just sends the pre-master secret, and then that pre-master secret is combined with the nonces of the server and the client. So, the pre-master secret and the nonces are combined, and then the master secret is generated from the PMS and nonces.

And then the master secret is used to generate the four keys. So, the main difference is that in the actual SSL, the nonces are part of the secret that is the master secret, so the nonces are part of the master secret, which is used to generate the secret keys E_B, E_A, M_B, and M_A. So, the inclusion of these nonces is a difference from the simplified SSL, and these nonces are used to defend against replay attacks as discussed. So, when the chosen symmetric key algorithm uses cipher block chaining, which we discussed earlier; recall that cipher block chaining requires some initialization vectors, one for each direction.

So, these initialization vectors are also obtained from the master secret. Typically, the data that is sent by the client to the server and the server to the client are long messages, and we discussed the encryption of long messages using symmetric key ciphers. So, one of the methods we discussed was cipher block chaining, which requires initialization vectors. So, these initialization vectors are also obtained from the master secret. Subsequently, all the messages that are sent between the client and server are encrypted, and a MAC is added to them.

So, the encryption is done using these secret keys E_B and E_A, and the MAC is generated using these secret keys M_B and M_A. So, this data exchange is quite similar to that in simplified SSL, which we discussed. So, the data flow is broken into records, and sequence numbers are used, and a MAC is added to every record. In step 5, the client sends a MAC of all the handshake messages it sent and received. And similarly, in step 6, the server sends a MAC of all the handshake messages it sent and received.

So, this MAC is, these MACs are computed over all these messages that were exchanged. And as we see, these MACs are meant for detecting any modification made by an intruder in these handshake messages. So, this is the actual SSL handshake consisting of these 6 steps. We now analyze this handshake. So, what is the reason for sending MACs of all the handshake messages in steps 5 and 6?

So, if we go back to this exchange, So, these MACs are sent in steps 5 and 6. So, we now discuss the reason for sending these MACs. So, these MACs are meant to detect any modification in the handshake messages by an intruder. One typical example of a modification is this.

Suppose Trudy controls a compromised intermediate router, then in step one, where the client sends a list of supported algorithms to the server, Trudy may delete the strong cryptographic algorithms from the list, which will force the server to select a weak one. So, for example, if AES, 3DES, and DES are in the list, then Trudy might delete the strong algorithm AES from the list. So, that will force the server to select a weak algorithm. So, any such modification by an intruder can be detected using these MACs that are included in the handshake in steps 5 and 6. So, these MACs are sent in steps 5 and 6, and they help to detect any modification in these first four messages.

And note that, why don't we send these MACs along with these first four messages themselves? So, the reason is that the keys have not yet been derived when these first four messages are sent. So, the secret keys are only derived in step 4. So, only after the secret keys are derived can the MACs be computed. So, for this reason, first the messages are sent, the handshake messages are sent, and then after the keys have been decided, the MACs are computed and sent for verifying if there has been any modification in the handshake messages.

If the server or the client detects an inconsistency in the MAC, that is, received in step 5 or 6, then the server or client terminates the connection. So, this means if there is any inconsistency in the MAC, then that means that an intruder has modified the messages, so

for this reason, the server or client terminates the connection. So, whoever detects the inconsistency terminates the connection. Now, what is the reason for using the nonces in steps 1 and 2? So, we see that in step 1, the client includes a client nonce in the message that it sends to the server, and in step 2, the server includes the server nonce to the client in the message that it sends to the client.

So, what's the reason for these nonces? So, the client nonce protects the client against the connection replay attack, and the server nonce protects the server against the connection replay attack. So, an example of the connection replay attack is this: an intruder, Trudy, may sniff all the messages of an SSL session, and the next day, Trudy may try to connect to Alice pretending to be Bob or vice versa. So, this is an example of a connection replay attack where all the messages exchanged in a connection are recorded, and then the intruder tries to communicate with one of the entities, either the server or the client, pretending to be the other entity. So, the client nonce protects the client against the connection replay attack, defends the client against attacks in which an intruder Trudy pretends to be the server and tries to connect to the client using recorded messages.

Similarly, the server nonce protects the server against the connection replay attack. It defends the server against attacks in which some intruder, Trudy, pretends to be the client and tries to connect to the server. Now, is server authentication done in this handshake? So, let's go back and see whether server authentication is done. So, we see that the question is, does the server prove its identity to the client in this handshake?

So, we see that at this step, the client sends a MAC of all the handshake messages that it has sent and received, and the server verifies it. Similarly, the server sends a MAC of all the handshake messages it has sent and received, and the client verifies it. So, server authentication is done because, in the second step, the server sends its certificate, and the client verifies it. And then, the client encrypts the pre-master secret using the public key in the certificate, and the client verifies the MAC in step 6. The client has sent the pre-master secret. And only if the server is legitimate will it be able to extract the premaster secret using its private key, which only the legitimate server has.

And using that private key, the server will extract the PMS and generate the secret keys, E_B, E_A, M_B, and M_A. The server will send the message authentication code to the client, and then the client verifies the message authentication code. So, that message authentication code will be correct only if the server has been able to extract the PMS and

derive the correct authentication key. So, using this process, server authentication is done in this handshake. Is client authentication done?

So, we see here that at no point does the client prove its identity to the server. So, client authentication is not done in this exchange. So, client authentication is not done. So, how is client authentication done in an SSL session? So, client authentication is optionally performed as part of an SSL handshake.

So, in that case, there is an additional step in the previous 6 steps. So, when client authentication is used as part of the handshake, a slightly different process is used with the addition of one step. We'll discuss that on the next slide. Often, client authentication is not performed in the SSL handshake because, after the handshake, the clients, who are often users such as Gmail account users, will share a password. Clients are often users, for example, Gmail account users, who do not have public-private key pairs or certificates.

So, after the handshake is completed, the client sends a password to the server. In particular, the server-side application using SSL prompts the client or user for a password, and then the client sends its password to the server, and the server verifies the password. So, using this process, the client is authenticated. So, recall that during our discussion of authentication, we discussed authentication protocols ap3.0 and ap3.1, which were vulnerable to the replay attack. In those protocols, the client also sent a password, and these protocols were vulnerable to the replay attack.

So, is this method of client authentication also vulnerable to the replay attack in SSL? So, the answer is no because the password is encrypted using a session key E_B, which is different for each SSL session. So, notice that the session key E_B is derived using the master secret, which is a function of the nonces, client nonce, and server nonce. So, hence, the session key is different for different SSL sessions. So, hence, this client authentication method is not vulnerable to the replay attack.

If an intruder records all the messages exchanged between the client and server and later on the intruder tries to authenticate as the client to the server, it won't be able to use the recorded messages because the nonces are different. So, the session key will be different in the next session. So, hence, this method is safe against the replay attack. Now, if client authentication is done as part of the SSL handshake, then the following steps are performed. After the server sends its own certificate to the client in step two, the server sends a certificate request message.

The client sends its certificate to the server, and the server extracts the public key of the client from the certificate. Next, the client sends a certificate verify message to the server, and this message contains the hash value of the handshake messages that are exchanged so far. This hash value is signed using the client's private key. So, this is a method for client authentication which can be used if the client has a certificate. Then the server applies the client's public key to the signed hash value and verifies the result to authenticate the client.

So, the question is, can an intruder later falsely authenticate itself as the client by replaying the certificate verify message? So, again, the answer is no because the hash value in the certificate verify message is a function of the server and client nonces, which are different for different SSL sessions. So, hence, this method is not vulnerable to the server and client nonce. They again defend against the replay attack. So, hence, this method for client authentication is safe against the replay attack.

So, we now discuss connection closure. After the client and server have exchanged all the data that they want to exchange, the connection can be closed. So, we now discuss how to securely close the connection. Suppose Bob wants to close the connection. One approach is that Bob sends a "TCP FIN" packet to close the underlying TCP connection.

So, a packet known as the FIN packet is used to close the TCP connection. So, one approach is that Bob directly sends a TCP FIN packet to close the underlying TCP connection on top of which SSL is running. So, is this a secure approach? This is not a secure approach because an intruder, Trudy, can close the connection before Bob has finished sending his data. So, this will prevent Bob from sending the remaining messages to the other side.

So, we need a secure procedure to close a connection. So, a procedure to close a connection in SSL is as follows. Bob sends an SSL record with the type field in it indicating that he wants to close the connection. So, recall that we discussed the format of an SSL record, and one of the fields in it is the type field, which indicates what kind of message it is. So, Bob sends an SSL record with the type field indicating that he wants to close the connection, and recall that the message authentication code is computed over different values, including the type field, so any modification in the type field can be detected by the receiver.

So, although the type field is sent in plain text form, it is included in the MAC computation, so any modification can be detected by the receiver. So, this is a procedure for securely closing the connection in SSL. Now recall that in steps one and two of the SSL handshake that we discussed, the client sends a list of cryptographic algorithms that it supports to the

server, and then the server selects a symmetric key algorithm, a public key algorithm, a MAC computation algorithm, and so on from this list and sends its choices. So, this is how the algorithms that will be used in the SSL session are negotiated. So, some examples of symmetric key algorithms that may be used in SSL are as follows:

- Examples of symmetric key algorithms:
  - DES, 3DES, DES40, AES, IDEA, RC4, RC2
- Examples of MAC computation algorithms:
  - MD5, SHA-1

DES, 3DES, DES40, AES, IDEA, RC4, RC2, and so on. So, these are different symmetric key algorithms that can be used. So, we see that possible symmetric key algorithms from this list, as well as some other algorithms, can be used in SSL. Some examples of MAC computation algorithms are MD5 and SHA-1. So, recall that a MAC is computed for a message m; the MAC is H(m,s). So, the hash function H can be computed either using MD5 or SHA-1 or some other hash functions as well, such as SHA-2, SHA-3, and so on.

Another part of the protocol that is exchanged is that the key exchange method used by the client and the server for agreeing upon the pre-master secret is also selected in steps 1 and 2 of the SSL handshake. The client and server have to agree upon the pre-master secret. What method do they use for agreeing upon the PMS? So, that method is used by, that is selected in steps one and two of the SSL handshake. So, we now discuss different key exchange methods.

Recall that the key exchange method used by the client and server for agreeing upon the PMS is selected in steps 1 and 2 of the SSL handshake. So, we discussed the 6-step SSL handshake above, and in step 3 of that handshake, the client generates a random pre-master secret, encrypts it with the server's RSA public key, and sends the encrypted PMS to the server. So, there the method for key exchange was that the client encrypts the PMS, and the encryption is done using the server's public key, and the encrypted PMS is sent to the server by the client. An alternative key exchange method is ephemeral Diffie-Hellman. So, this is more secure than this method.

So, we'll discuss the sense in which it is more secure. So, we have discussed the Diffie-Hellman protocol in a previous lecture. So, this is a variant known as ephemeral Diffie-Hellman, where public keys are used in the Diffie-Hellman exchange. So, in this method, the client and server randomly generate one-time Diffie-Hellman private and public keys.

So, we discussed that the private key is SA for Alice, and the public key is TA in the notation that we used in our discussion of Diffie-Hellman.

So, the client and server randomly choose one-time Diffie-Hellman private and public keys. The Diffie-Hellman public key is signed using the RSA private key of the sender and sent to the receiver. So, this is used for message integrity. So, the public key needs to be shared from the sender to the receiver. So, it is signed using the RSA private key so that the receiver can detect any modification in the public key that is in the Diffie-Hellman public key sent by the sender.

Then, using the received values, the client and server compute the shared secret, as in Diffie-Hellman, and this shared secret acts as the pre-master secret. Then, using this pre-master secret and the nonces, the master secret is generated. So, why is the Diffie-Hellman public key signed using the RSA private key of the sender? So, the reason is to prevent the man-in-the-middle attack. In our discussion of Diffie-Hellman, we discussed the man-in-the-middle attack on Diffie-Hellman, where an intruder can modify the public key sent by the sender and send their own public key to the receiver.

So, to prevent such attacks, the public key needs to be, we need to add message integrity to the Diffie-Hellman public key. So, that message integrity is added by signing the Diffie-Hellman public key using the RSA private key of the sender. So, we see that this is quite a complicated method where the Diffie-Hellman algorithm needs to be used, and then the public keys need to be exchanged, and from the public keys that are exchanged, the shared secret is derived. So, whereas the previous method was quite simple, where the client simply generated a random PMS, encrypted it, and sent it to the server. An advantage of this method, ephemeral Diffie-Hellman, over the method in which the client generates a random PMS and sends the PMS encrypted using the server's RSA public key is as follows:

Consider an eavesdropping intruder who records all the messages exchanged between the client and server. So, all the SSL messages exchanged are recorded by the eavesdropping intruder. And later on, the intruder is somehow able to obtain the RSA private key of the server. So, sometime in the future, the intruder is somehow able to obtain the RSA private keys. So, then the intruder can go back to this recorded session and decrypt the messages.

Random PMS encrypted using the server's RSA public key to the server. So, in this latter method, the intruder can decrypt all the communication that took place between the client and server. That's because once the private key is leaked, the intruder can decrypt the encrypted PMS and use that PMS to derive the master secret because the nonces have been

sent without any encryption in steps 1 and 2. So, the intruder who has recorded all the messages can use the nonces and the pre-master secret which the intruder has decrypted to generate the master secret, and this master secret can be used to derive the keys E_B, E_A, M_B, and MA, and then the intruder can decrypt all the communication that took place between the client and server. But under the ephemeral Diffie-Hellman method, the intruder cannot decrypt the communication because the pre-master secret cannot be computed using messages exchanged between the client and the server.

Recall that Diffie-Hellman has the property that if there is an eavesdropping intruder who records all the messages that are sent between the client and server, even then, the intruder is not able to derive the shared key between the client and server. So, the shared key in our notation that we used was g to the power SASB mod p. So, the shared key was this: g to the power SASB mod p in the notation that we used in our discussion of Diffie-Hellman. So, this shared key cannot be derived even if an intruder sniffs on the exchange and obtains the public keys TA and TB. So, the secret key cannot be obtained from TA and TB, so hence, under the ephemeral Diffie-Hellman method, the intruder cannot decrypt the communication even if the intruder gets the private key of the server sometime in the future. So, the ephemeral Diffie-Hellman method provides forward secrecy, whereas the latter method does not.

By forward secrecy, we mean that if the long-term key, such as the private key, is compromised sometime in the future, then the intruder is able to go back to the recorded session and decrypt the messages that were exchanged. If the intruder is not able to decrypt the messages sometime in the future, then the method is said to provide forward secrecy. So, the ephemeral Diffie-Hellman method provides forward secrecy, whereas the latter method does not provide forward secrecy. In this sense, the ephemeral Diffie-Hellman method is more secure than the simple method. The PMS is simply encrypted and sent using the server's public key.

In summary, we discussed SSL and TLS. First, we discussed simplified SSL, and then we discussed the actual SSL. We discussed that during the handshake, different algorithms are negotiated, and we discussed key exchange methods that can be used, including ephemeral Diffie-Hellman, which is a secure key exchange method. This concludes our discussion of SSL. Thank you.