

CYB650

# Introduction to Cryptography

## *Transport-Level Security*



Prepared by Dr. Esraa Saleh Alomari  
Lecture #6



جامعة الجميع الذكية  
EVERYONE'S SMART UNIVERSITY

*The Part 2 of the module 3 covers the Transport-Level Security are as follows:*

❑ **Transport Layer Security**

- **Version Number**
- **Message Authentication Code**
- **Pseudorandom Function**
- **Alert Codes**
- **Cipher Suites**
- **Client Certificate Types**
- **Web Traffic Security**
- **Certificate-Verify and Finished Messages**

❑ **HTTPS**

- **Connection Initiation**
- **Connection Closure**

❑ **Secure Shell (SSH)**

- **Transport Layer Protocol**
- **User Authentication Protocol**
- **Connection Protocol**



# Transport Layer Security

- ❑ TLS is an IETF standardization initiative whose goal is to produce an Internet standard version of SSL. TLS is defined as a Proposed Internet Standard in RFC 5246. RFC 5246 is very similar to SSLv3. In this section, we highlight the differences.

- ❑ The TLS Record Format is the same as that of the SSL Record Format (Figure 17.4), and the fields in the header have the same meanings. The one difference is in version values. For the current version of TLS, the major version is 3 and the minor version is 3.

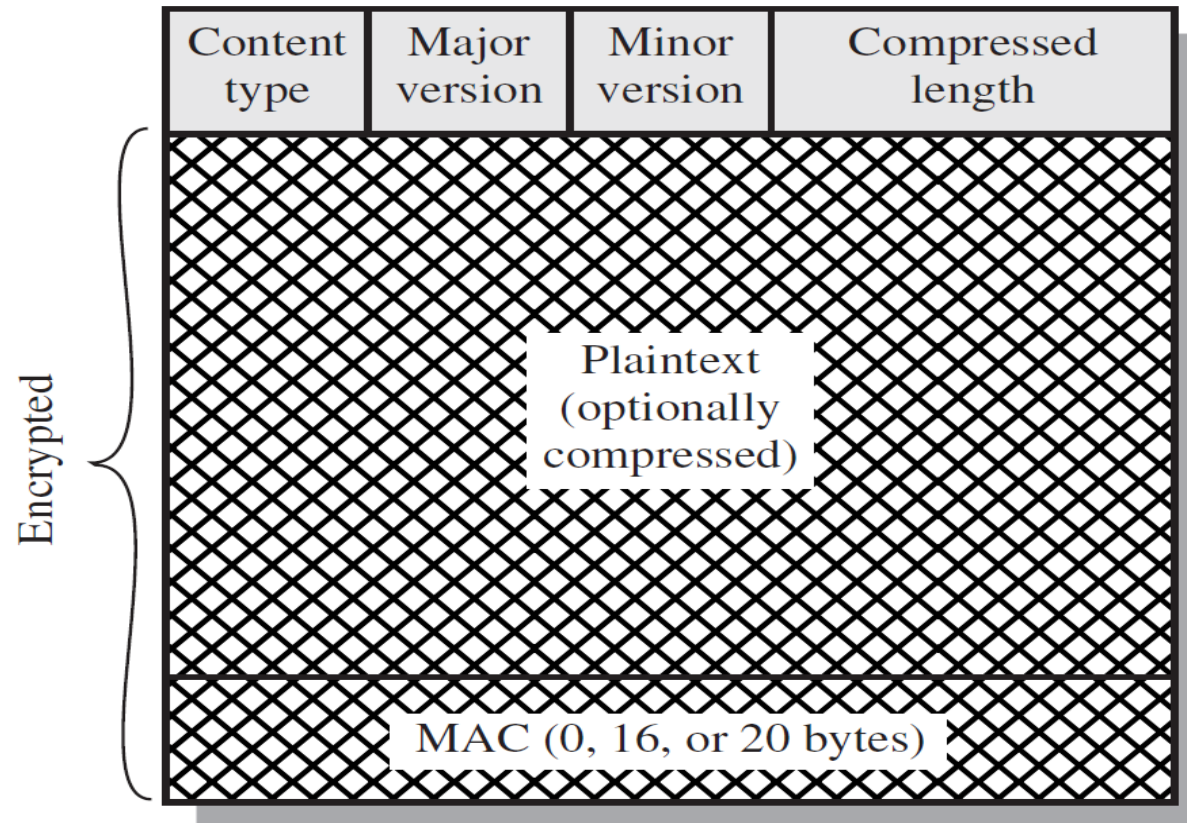


Figure 17.4 SSL Record Format

- ❑ There are two differences between the SSLv3 and TLS MAC schemes: the actual algorithm and the scope of the MAC calculation. TLS makes use of the HMAC algorithm defined in RFC 2104.

## Message Authentication Code

$$\text{HMAC}_K(M) = H[(K^+ \oplus \text{opad}) \parallel H[(K^+ \oplus \text{ipad}) \parallel M]]$$

where

$H$  = embedded hash function (for TLS, either MD5 or SHA-1)

$M$  = message input to HMAC

$K^+$  = secret key padded with zeros on the left so that the result is equal to the block length of the hash code (for MD5 and SHA-1, block length = 512 bits)

$\text{ipad}$  = 00110110 (36 in hexadecimal) repeated 64 times (512 bits)

$\text{opad}$  = 01011100 (5C in hexadecimal) repeated 64 times (512 bits)

- ❑ SSLv3 uses the same algorithm, except that the padding bytes are concatenated with the secret key rather than being XORed with the secret key padded to the block length. The level of security should be about the same in both cases.
- ❑ For TLS, the MAC calculation encompasses the fields indicated in the following expression:

```
MAC (MAC_write_secret, seq_num || TLSCompressed.type ||  
      TLSCompressed.version || TLSCompressed.length ||  
      TLSCompressed.fragment)
```

- ❑ The MAC calculation covers all of the fields covered by the SSLv3 calculation, plus the field TLS Compressed. version, which is the version of the protocol being employed.

- ❑ TLS makes use of a pseudorandom function referred to as PRF to expand secrets into blocks of data for purposes of key generation or validation. The objective is to make use of a relatively small shared secret value but to generate longer blocks of

# Pseudorandom Function

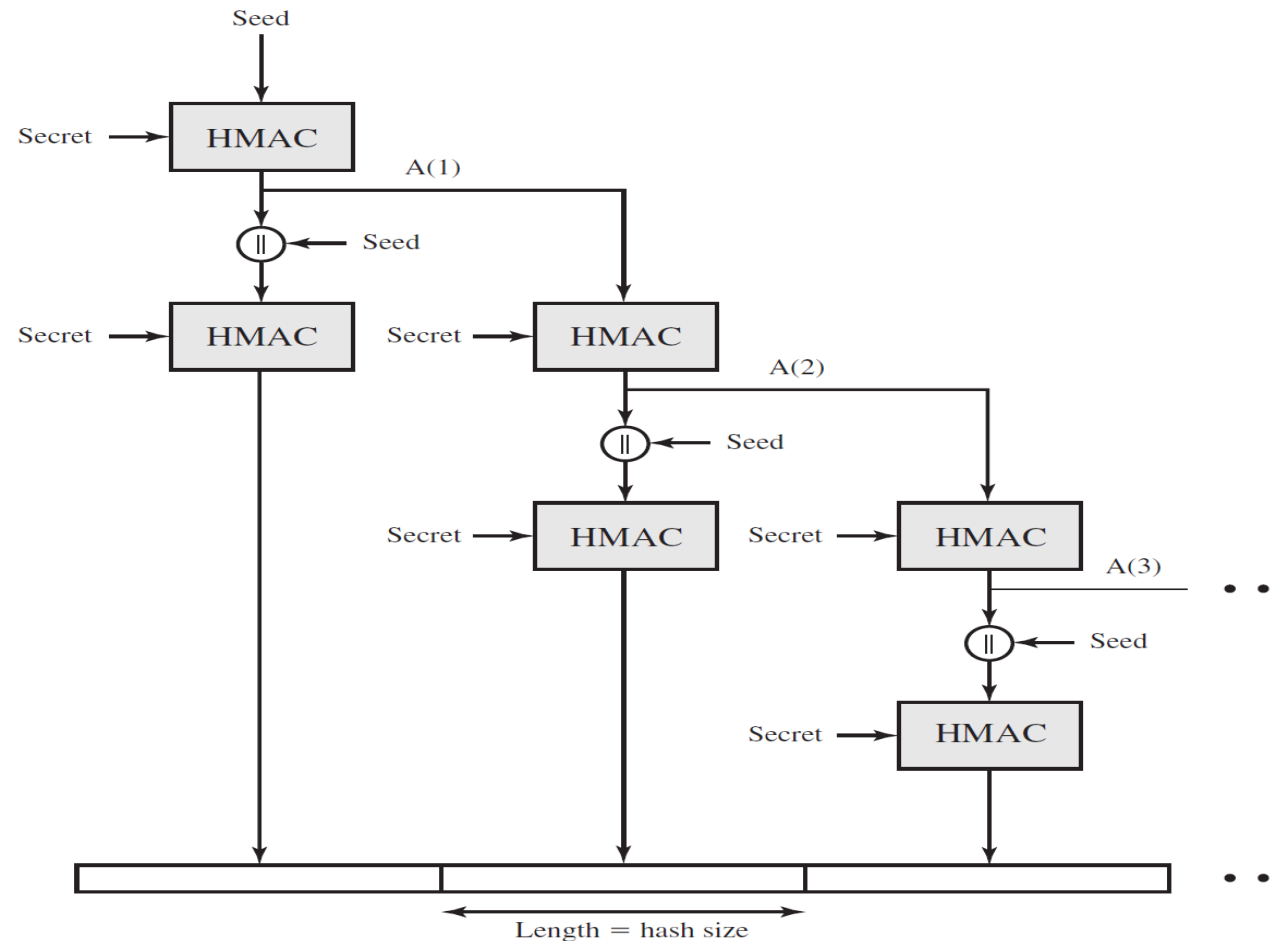


Figure 17.7 TLS Function  $P\_hash(secret, seed)$

data in a way that is secure from the kinds of attacks made on hash functions and MACs. The PRF is based on the data expansion function (Figure 17.7) given as

$$\begin{aligned} P\_hash(secret, seed) = & HMAC\_hash(secret, A(1) \parallel seed) \parallel \\ & HMAC\_hash(secret, A(2) \parallel seed) \parallel \\ & HMAC\_hash(secret, A(3) \parallel seed) \parallel \dots \end{aligned}$$

where  $A()$  is defined as

$$\begin{aligned} A(0) &= seed \\ A(i) &= HMAC\_hash(secret, A(i-1)) \end{aligned}$$

- ❑ The data expansion function makes use of the HMAC algorithm with either MD5 or SHA-1 as the underlying hash function. As can be seen, P\_hash can be iterated as many times as necessary to produce the required quantity of data. For example, if P\_SHA-1 was used to generate 64 bytes of data, it would have to be iterated four times, producing 80 bytes of data of which the last 16 would be discarded

## Pseudorandom Function

- ❑ In this case, P\_MD5 would also have to be iterated four times, producing exactly 64 bytes of data. Note that each iteration involves two executions of HMAC—each of which in turn involves two executions of the underlying hash algorithm.
- ❑ To make PRF as secure as possible, it uses two hash algorithms in a way that should guarantee its security if either algorithm remains secure. PRF is defined as

$$\text{PRF}(\text{secret}, \text{label}, \text{seed}) = \text{P\_hash}(\text{S1}, \text{label} \} \text{seed})$$

- ❑ PRF takes as input a secret value, an identifying label, and a seed value and produces an output of arbitrary length.

- ❑ TLS supports all of the alert codes defined in SSLv3 with the exception of `no_certificate`. A number of additional codes are defined in TLS; of these, the following are always fatal.
- ❑ **record\_overflow**: A TLS record was received with a payload (ciphertext) whose length exceeds  $2^{14} + 2048$  bytes, or the ciphertext decrypted to a length of greater than  $2^{14} + 1024$  bytes.

- ❑ **unknown\_ca:** A valid certificate chain or partial chain was received, but the certificate was not accepted because the CA certificate could not be located or could not be matched with a known, trusted CA.
- ❑ **access\_denied:** A valid certificate was received, but when access control was applied, the sender decided not to proceed with the negotiation.
- ❑ **decode\_error:** A message could not be decoded, because either a field was out of its specified range or the length of the message was incorrect.

- ❑ **protocol\_version:** The protocol version the client attempted to negotiate is recognized but not supported.
- ❑ **insufficient\_security:** Returned instead of `handshake_failure` when a negotiation has failed specifically because the server requires ciphers more secure than those supported by the client.
- ❑ **unsupported\_extension:** Sent by clients that receive an extended server hello containing an extension not in the corresponding client hello.

- ❑ **internal\_error**: An internal error unrelated to the peer or the correctness of the protocol makes it impossible to continue.
- ❑ **decrypt\_error**: A handshake cryptographic operation failed, including being unable to verify a signature, decrypt a key exchange, or validate a finished message.

- ❑ There are several small differences between the cipher suites available under SSLv3 and under TLS:
- ❑ **Key Exchange:** TLS supports all of the key exchange techniques of SSLv3 with the exception of Fortezza.
- ❑ **Symmetric Encryption Algorithms:** TLS includes all of the symmetric encryption algorithms found in SSLv3, with the exception of Fortezza.

## Client Certificate Types

- ❑ TLS defines the following certificate types to be requested in a `certificate_request` message: `rsa_sign`, `dss_sign`, `rsa_fixed_dh`, and `dss_fixed_dh`. These are all defined in SSLv3. In addition, SSLv3 includes `rsa_ephemeral_dh`, `dss_ephemeral_dh`, and `fortezza_kek`. Ephemeral Diffie-Hellman involves signing the Diffie-Hellman parameters with either RSA or DSS. For TLS, the `rsa_sign` and `dss_sign` types are used for that function; a separate signing type is not needed to sign Diffie-Hellman parameters. TLS does not include the Fortezza scheme.

- ❑ In the TLS `certificate_verify` message, the MD5 and SHA-1 hashes are calculated only over `handshake_messages`. Recall that for SSLv3, the hash calculation also included the master secret and pads. These extra fields were felt to add no additional security.
- ❑ As with the finished message in SSLv3, the finished message in TLS is a hash based on the shared `master_secret`, the previous handshake messages, and a label that identifies client or server.

❑ The calculation is somewhat different. For TLS, we have

$$\text{PRF}(\text{master\_secret}, \text{finished\_label}, \text{MD5}(\text{handshake\_messages}) \parallel \text{SHA-1}(\text{handshake\_messages}))$$

where finished\_label is the string “client finished” for the client and “server finished” for the server.

- ❑ The pre\_master\_secret for TLS is calculated in the same way as in SSLv3. As in SSLv3, the master\_secret in TLS is calculated as a hash function of the pre\_master\_secret and the two hello random numbers. The form of the TLS calculation is different from that of SSLv3 and is defined as

$$\text{master\_secret} = \text{PRF}(\text{pre\_master\_secret}, \text{"master secret"}, \text{ClientHello.random } \{ \\ \text{ServerHello.random})$$

The algorithm is performed until 48 bytes of pseudorandom output are produced.

The calculation of the key block material (MAC secret keys, session encryption keys, and IVs) is defined as

```
key_block = PRF(master_secret, "key expansion",  
                SecurityParameters.server_random ||  
                SecurityParameters.client_random)
```

until enough output has been generated. As with SSLv3, the `key_block` is a function of the `master_secret` and the client and server random numbers, but for TLS, the actual algorithm is different.

- ❑ In SSL, the padding added prior to encryption of user data is the minimum amount required so that the total size of the data to be encrypted is a multiple of the cipher's block length. In TLS, the padding can be any amount that results in a total that is a multiple of the cipher's block length, up to a maximum of 255 bytes. For example, if the plaintext (or compressed text if compression is used) plus MAC plus padding. length byte is 79 bytes long, then the padding length (in bytes) can be 1, 9, 17, and so on, up to 249. A variable padding length

may be used to frustrate attacks based on an analysis of the lengths of exchanged messages.

- ❑ HTTPS (HTTP over SSL) refers to the combination of HTTP and SSL to implement secure communication between a Web browser and a Web server. The HTTPS capability is built into all modern Web browsers. Its use depends on the Web server supporting HTTPS communication. For example, some search engines do not support HTTPS. Google provides HTTPS as an option: <https://google.com>.

❑ The principal difference seen by a user of a Web browser is that URL (uniform resource locator) addresses begin with https:// rather than http://. A normal HTTP connection uses port 80. If HTTPS is specified, port 443 is used, which invokes SSL.

❑ When HTTPS is used, the following elements of the communication are

encrypted:

- ☐ URL of the requested document
  - ☐ Contents of the document
  - ☐ Contents of browser forms (filled in by browser user)
  - ☐ Cookies sent from browser to server and from server to browser
  - ☐ Contents of HTTP header
- ☐ HTTPS is documented in RFC 2818, HTTP Over TLS. There is no fundamental change in using HTTP over either SSL or TLS, and both implementations are referred to as HTTPS.

- ❑ For HTTPS, the agent acting as the HTTP client also acts as the TLS client. The client initiates a connection to the server on the appropriate port and then sends the TLS ClientHello to begin the TLS handshake. When the TLS handshake has finished, the client may then initiate the first HTTP request. All HTTP data is to be sent as TLS application data. Normal HTTP behavior, including retained connections, should be followed.

- ❑ There are three levels of awareness of a connection in HTTPS. At the HTTP level, an HTTP client requests a connection to an HTTP server by sending a connection request to the next lowest layer. Typically, the next lowest layer is TCP, but it also may be TLS/SSL.

- ❑ At the level of TLS, a session is established between a TLS client and a TLS server. This session can support one or more connections at any time. As we have seen, a TLS request to establish a connection begins with the establishment of a TCP connection between the TCP entity on the client side and the TCP entity on the server side.

- ❑ An HTTP client or server can indicate the closing of a connection by including the following line in an HTTP record: Connection: close. This indicates that the connection will be closed after this record is delivered. The closure of an HTTPS connection requires that TLS close the connection with the peer TLS entity on the remote side, which will involve closing the underlying TCP connection. At the TLS level, the proper way to close a connection is for each side to use the TLS alert protocol to send a close\_notify alert. TLS implementations must initiate an exchange of closure alerts before closing a connection.

- ❑ A TLS implementation may, after sending a closure alert, close the connection without waiting for the peer to send its closure alert, generating an “incomplete close”. Note that an implementation that does this may choose to reuse the session. This should only be done when the application knows (typically through detecting HTTP message boundaries) that it has received all the message data that it cares about.

- ❑ HTTP clients also must be able to cope with a situation in which the underlying TCP connection is terminated without a prior close\_notify alert and without a Connection: close indicator. Such a situation could be due to a programming error on the server or a communication error that causes the TCP connection to drop. However, the unannounced TCP closure could be evidence of some sort of attack. So the HTTPS client should issue some sort of security warning when this occurs.

- ❑ Secure Shell (SSH) is a protocol for secure network communications designed to be relatively simple and inexpensive to implement. The initial version, SSH1 was focused on providing a secure remote logon facility to replace TELNET and other remote logon schemes that provided no security. SSH also provides a more

## Secure Shell (SSL)

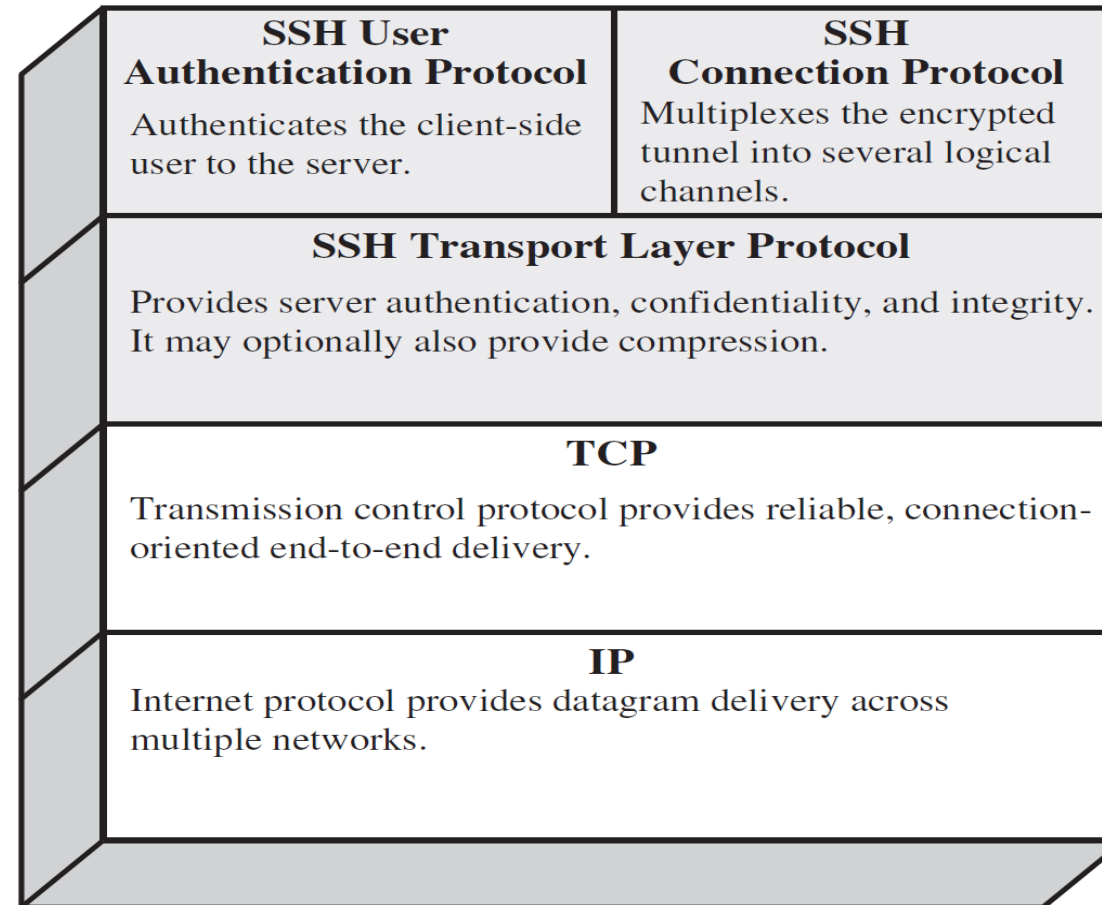


Figure 17.8 SSH Protocol Stack

❑ General client/server capability and can be used for such network functions as file transfer and e-mail. A new version, SSH2, fixes a number of security flaws in the original scheme. SSH2 is documented as a proposed standard in IETF RFCs 4250 through 4256.

❑ SSH client and server applications are widely available for most operating systems. It has become the method of choice for remote login and X tunneling and is rapidly becoming one of the most pervasive applications for encryption technology outside of embedded systems.

- ❑ SSH is organized as three protocols that typically run on top of TCP (Figure 17.8):
- ❑ **Transport Layer Protocol:** Provides server authentication, data confidentiality, and data integrity with forward secrecy (i.e., if a key is compromised during one session, the knowledge does not affect the security of earlier sessions). The transport layer may optionally provide compression.

- ❑ **User Authentication Protocol:** Authenticates the user to the server.
- ❑ **Connection Protocol:** Multiplexes multiple logical communications channels over a single, underlying SSH connection.

# Thank You

ANY  
QUESTIONS?

