16.3 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.

Version Number

The TLS Record Format is the same as that of the SSL Record Format (Figure 16.4), 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.

Message Authentication Code

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. Recall from Chapter 12 that HMAC is defined as

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

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)
ipad	=	00110110 (36 in hexadecimal) repeated 64 times (512 bits)
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 `TLSCompressed.version`, which is the version of the protocol being employed.

Pseudorandom Function

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 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 16.7) given as

```
P_hash(secret, seed) = HMAC_hash(secret,A(1) ‖ seed) ‖
                       HMAC_hash(secret, A(2) ‖ seed) ‖
                       HMAC_hash(secret, A(3) ‖ seed) ‖ . . .
```

where `A()` is defined as

```
A(0) = seed
A(i) = HMAC_hash(secret, A(i – 1))
```

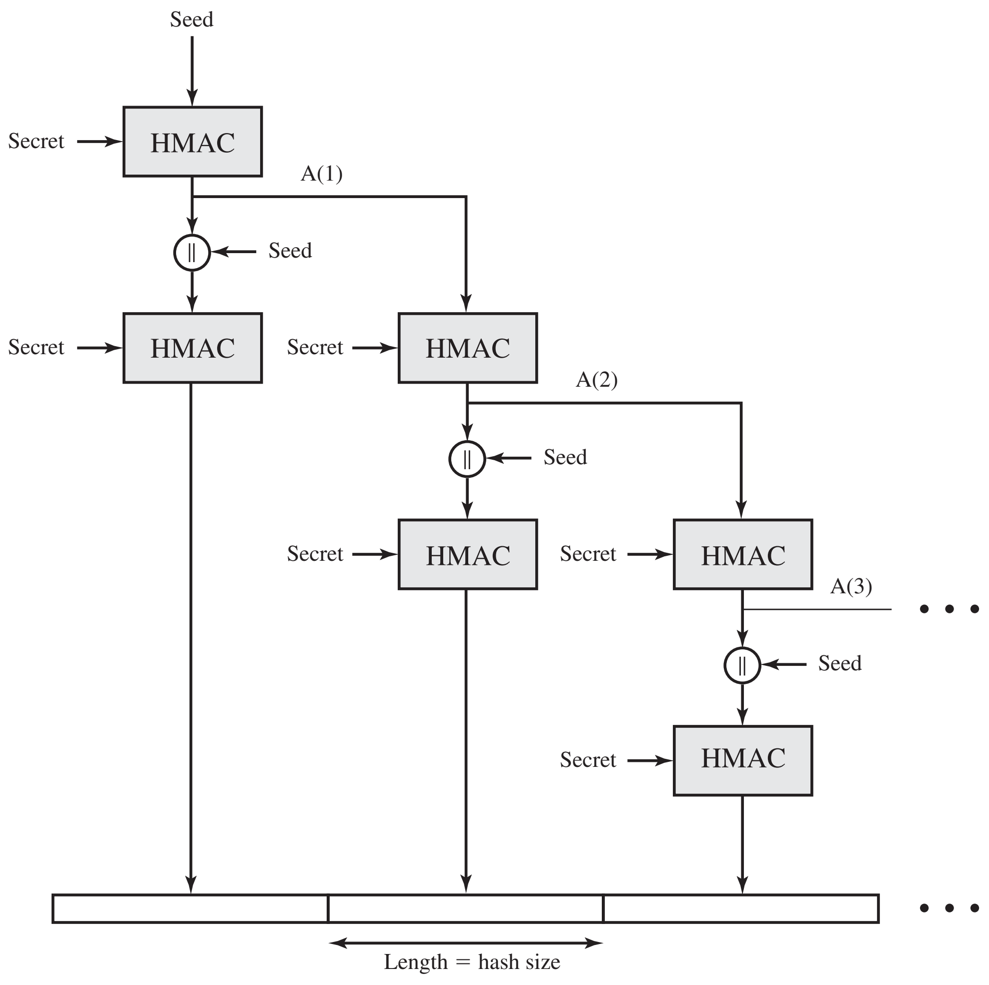

Figure 16.7 TLS Function `P_hash(secret, seed)`

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. 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

```
PRF(secret, label, seed) = P_hash(S1,label || seed)
```

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

Alert Codes

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.

The remaining alerts include the following.

- **`user_canceled`:** This handshake is being canceled for some reason unrelated to a protocol failure.
- **`no_renegotiation`:** Sent by a client in response to a hello request or by the server in response to a client hello after initial handshaking. Either of these messages would normally result in renegotiation, but this alert indicates that the sender is not able to renegotiate. This message is always a warning.

Cipher Suites

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_kea`. 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.

`certificate_verify` and Finished Messages

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

```
PRF(master_secret,finished_label,MD5(handshake_messages)‖
    SHA-1(handshake_messages))
```

where `finished_label` is the string "client finished" for the client and "server finished" for the server.

Cryptographic Computations

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

```
master_secret= PRF(pre_master_secret,"master secret",
               ClientHello.random || 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.

Padding

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.

16.4 HTTPS

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, search engines do not support HTTPS.

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.