Notes on Transport Layer Security

Vahab Pournaghshband
Computer Science Department
University of California, Los Angeles
vahab@cs.ucla.edu

Abstract

This note provides a brief overview of Transport Layer Security (TLS) protocol version 1.2 which provides security for communications on the Internet. TLS, similar to its successor SSL, allows client/server applications to communicate in a way that is designed to prevent eavesdropping,tampering, or message forgery for secure communication on the Internet[1]. Block cipher and keyed-Hash Message Authentication Code are first introduced prior to the discussion on the details of the two main components of TLS, Handshake and Record protocols.

1 Background

1.1 Block Cipher

Block cipher is a symmetric key encryption algorithm which uses the idea of encrypting blocks of letters in a message as opposed to encrypting individual letters independently.

1.1.1 Electronic Codebook (ECB)

ECB is the simplest type of block cipher which first breaks the plaintext, P, into L pieces $P = [P_1, P_2, ..., P_L]$ and then encrypts each individual block independently using a keyed-function: $C_i = E_K(P_i)$. The ciphertext is the concatenation of all encrypted blocks, $C = [C_1, C_2, ..., C_L]$.

1.1.2 Cipher Block Chaining (CBC)

Unlike ECB, CBC block cipher does not encrypt individual blocks independently. To introduce some dependency, CBC uses a feedback mechanism that involves earlier encrypted blocks into encrypting subsequent blocks:

$$C_j = E_K(P_j \oplus C_j - 1)$$

Where $\oplus$ is the binary XOR operator, and C_0 is called the initialization vector and is preselected.

1.2 HMAC-based Pseudorandom Function

Message Authentication Code (MAC) is a piece of information used for both authentication and integrity checking of a message. A MAC algorithm is a key-based algorithm that produces the MAC. The MAC algorithm is a deterministic algorithm, meaning that it always produces the same MAC for a given message. A keyed-Hash Message Authentication Code (HMAC), specifically employs an iterative cryptographic hash function in combination with a secret key. HMAC with a secret key, k, and an iterative hash function, h, is computed as follows:

$$HMAC_{k,h}(m) = h((k \oplus opad)||h((k \oplus ipad)||m))$$

where *opad* and *ipad* are outer and inner padding with constant hexadecimal values of 0x5C5C...5C and 0x3636...36 respectively. SHA-256 is widely used in practice as the HMAC hash function in TLS current implementations.

Hash functions in TLS are used to generate a pseudorandom number known to both parties but constructed independently as a mean to agree on an arbitrarily large shared private key. HMAC, however, produces a fixed length message digest which may not be sufficiently long. To remedy this, successive concatenation of HMAC with different values is done. Formally we define P_{hash}, a hash function with aribitrary long message digest, as:

$$\begin{aligned} P_{hash}(k, s) = {} & HMAC_{k,h}(A_1||s)|| \\ & HMAC_{k,h}(A_2||s)|| \\ & HMAC_{k,h}(A_3||s)|| \\ & \dots \end{aligned}$$

Where k is the secret key, s is the seed, $A_i = \begin{cases} s & i = 0 \\ HMAC_{k,h}(A_{i-1}) & i \neq 0 \end{cases}$, and $||$ indicates concatenation.
In TLS terminology, Pseudorandom Function (PRF) is designed to generate shared private keys. The TLS pseudorandom function takes a secret key k, seed s, and an identifying label denoted as L in the definition, and produces an output of arbitrary length. PRF in the TLS specification is defined as follows:

$$PRF(k, L, s) = P_{hash}(k, L||s)$$

The label, L, in the pseudorandom function is a numerical representation of an ASCII string without the trailing null character.

2 Transport Layer Security

Transport Layer Security (TLS) is a cryptographic protocol that is designed to provide both security and data integrity for communications over a reliable transport protocol such as Transport Control Protocol (TCP). TLS allows client-server applications to communicate across a public network while preventing eavesdropping, tampering, and message forgery by providing endpoint authentication and confidentiality over the Internet. The goals of the TLS protocol, in order of priority, are cryptographic security, interoperability, extensibility, and relative efficiency. TLS is designed to be application protocol independent. TLS protocol consists of two main components: Handshake protocol, to set session states and shared private keys, and Record protocol, to transmit data securely using the shared keys.

2.1 Handshake Protocol

The cryptographic parameters of the session state are produced by the Handshake Protocol. The TLS Record Protocol will then resume the connection between the two parties. In the Handshake protocol, both parties acknowledge their protocol versions, agree on cryptographic and compression algorithms, optionally authenticate each other through certificates, and use public-key encryption techniques to generate shared private keys. A step-by-step overview of what and how client and server communicate in the Handshake protocol is presented below.

1. Client sends a message publicly containing the following information:

   - The highest version of TLS the client's computer can support
   - 32-byte random number r_A consisting of a 4-byte timestamp and a 28-byte random number [1]
   - A Cipher Suite list in decreasing order of preference for each of the following algorithm families: Public-Key Algorithm (PKA), encryption algorithm used in the Cipher Block Chaining, and compression algorithm (COMPRESS).[2]

2. Server informs the client about the winning algorithms (after examining the Cipher Suite list sent by the client) along with a 32-byte random number r_B constructed similarly as r_A.

3. Client replies with a number called pre-master secret s_{pm} using the public key algorithm PKA with public keys retrieved from the server's certificate signed by a Certifying Authority (CA).

4. Both parties independently calculate the 48-byte long master secret, s_m, to further obtain the keys to exchange data. The master secret is calculated using PRF:

   $$s_m = PRF(s_{pm}, "master\ secret", r_A||r_B)$$

   It is worth mentioning that in the previous version of TLS the master secret was computed as follows, before MD5 proven to be insecure[4]:

   $$\begin{gathered}\text{MD5}(s_{pm}||\text{SHA-1}(A||s_{pm}||r_A||r_B)) \quad || \\ \text{MD5}(s_{pm}||\text{SHA-1}(BB||s_{pm}||r_A||r_B)) \quad || \\ \text{MD5}(s_{pm}||\text{SHA-1}(CCC||s_{pm}||r_A||r_B))\end{gathered}$$

   Where A, BB, and CCC are strings added for padding.

[1] The timestamp used in generating r_A will be used to avoid reply attack which is an attempt by attacker to reuse a used session state.

[2] Since there are various encryption algorithms with different levels of security and computation time, it is legitimate to leave this as an option to the parties.

5. At this stage, both parties know s_m, s_{pm}, r_A, and r_B. they independently compute the Key Block (KB) that contains all needed private shared keys for this session:

$$KB = PRF(s_m, "key\ expansion", r_A||r_B)$$

KB is then broken into six pieces and labeled as K_1, K_2, ..., and K_6, before terminating the Handshake phase.

2.2 Record Protocol

Now the client and the server are ready to communicate securely using the key block as a set of security parameters obtained by the Handshake protocol. The Record protocol takes data to be transmitted in one endpoint, fragments the data into manageable blocks, compresses the data, applies a MAC, encrypts by block cipher, and transmits the result. Received data is then decrypted, verified, decompressed, reassembled, and then delivered to higher-level application on the other endpoint. In short, Record protocol ensures that the connection is private via symmetric encryption by session-unique keys and reliable via integrity check. Suppose the client wants to send data chunk, d. The client:

1. Compresses the data using the agreed algorithm: $d' = COMPRESS(d)$
2. Hashes the compressed data for data integrity using K_2: $d'' = \{d', HMAC_{K_2}(d')\}$
3. Encrypts the data along with its MAC using CBC mode block cipher BCA where the secret key is K_1 and the initialization vector is K_3: $d''' = BCA_{K1}(d'', K_3)$
4. Sends d''' over the public channel

And the server retrieves d from d''':

1. Decrypts the data along with its MAC using BCA_{K1}.
2. Verifies data integrity by computing HMAC of data using K_2 and comparing it with the HMAC computed on the client side
3. Decompresses to retrieve d

The process is reveresed when server wants to send data to the client while the last three pieces of the key block is used instead.

References

[1] Dierks, T. and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246, August 2008.

[2] Dierks, T. and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.1*, RFC 4346, April 2006.

[3] Krawczyk, H., Bellare, M., and R. Canetti, *HMAC: Keyed- Hashing for Message Authentication*, RFC 2104, February 1997.

[4] Marc Stevens, Arjen Lenstra, Benne de Weger, *Vulnerability of software integrity and code signing applications to chosen-prefix collisions for MD5*, Nov 30, 2007.