International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-3 Number-3 Issue-12 September-2013

Implementation of OpenSSL API's for TLS 1.2 Operation

Maria Navin J R[1], Suresh P[2], Pradeep K R[3]

Abstract

OpenSSL is a popular and effective open source version of SSL/TLS, the most widely used protocol for secure network communications to provide data unity and secrecy between two different applications. TLS 1.2 is currently the most secure and up to date version of the standard. The main advantages of TLS 1.2 over previous versions are: It fixes cryptography flaws over previous version, supports additional crypto algorithms and supports flexibility in defining those algorithms. Fixing of flaws increases security which can be achieved by modifying the protocol to meet the better requirement and constraints for security needed. OpenSSL is a high quality package used by many of the commercial products available in the market. Achieving RFC compliance is very important for commercial products, as OpenSSL does not support TLS 1.2, the main objective of this paper is to show how TLS 1.2 can be supported in OpenSSL.

Keywords

Message Integrity, Public Key Cryptography, OpenSSL, Symmetric Cryptography, TLS.

1. Introduction

Data integrity and confidentiality between communicating applications can be achieved using the TLS protocol. The protocol is stacked with two layers: namely the TLS Record and Handshake protocols. These protocols are, layered on top of some reliable transport protocol (e.g., TCP [1]). The TLS Record Protocol provides security while making connection which comprises of two basic properties:

This work was supported in part by the Department of CSE & ISE, SVCE, Bangalore.

Maria Navin J R, Assistant Professor, Department of ISE, SVCE, Bangalore.
Suresh P, Assistant Professor, Department of CSE, SVCE, Bangalore, India.
Pradeep K R, Assistant Professor, Department of ISE, SVCE, Bangalore, India.

The connection established is confidential and encryption of data is done using symmetric cryptography (e.g., AES [2], RC4 [3], etc.). The keys for this symmetric encryption are generated uniquely for each connection and are based on a secret key negotiated by another protocol [4].

- The connection is reliable. The transport of message includes a message integrity check using a keyed Medium Access Control (MAC). Secure hash functions (e.g., SHA-1, etc.) are used for computations of MAC [5].

The TLS Record Protocol can operate without a MAC, but is generally only used in message integrity check mode while another protocol are working with the help of Record Protocol as a transport for negotiating security parameters. The TLS Record Protocol can be used by encapsulating the various higher-level protocols. The protocol allows negotiating an encryption algorithm and cryptographic keys between the server and the client to authenticate each other before the application protocol transmits or receives its first byte of data.

The TLS Protocol provides security while connecting which has three basic properties:
- The identity of the peer's can be authenticated using public key or asymmetric key cryptography (e.g., RSA [6], DSA [7], etc.). The authentication can be made optional, but while communicating between peer to peer one of the peer needs it.
- The negotiation is secured by having secret key, the negotiated secret key is unavailable to eaves droppers and for any connection which is authenticated, the secret key cannot be obtained even by an attacker who can place himself in the middle of the connection.
- The negotiation made between the peer's is reliable, where no attacker can modify the negotiation made between the peer's without being detected by the parties while making the communication.

The advantage of using TLS is since it is independent application protocol and higher-level protocols can be placed on top of the TLS protocol transparently. But the TLS standard does not specify how the security is added to protocol. The decisions made on to initiate TLS handshaking and interpretation of the exchanging the authentication certificates are left to the judgment of the designers and protocols implementers which will run on top of TLS.

2. TLS 1.2 comparison with TLS 1.1

TLS 1.2 [6] protocol contains improved flexibility, particularly in negotiation of cryptography algorithms. Major differences compared with TLS 1.1 [7] are:

- The MD5-SHA-1 combination in the pseudorandom function (PRF) was replaced with SHA-256, with an option to use cipher-suite specified PRFs.
- Cipher-Suite specific hash algorithms as an option were introduced in SHA-256 which replaced MD5-SHA-1 in the finished message.
- The MD5-SHA-1 combination in the digitally-signed element was replaced with a single hash. Signed elements now include a field that explicitly specifies the hash algorithm used.
- Specification of hash and signature algorithms by the clients and servers.
- Expansion of support for authenticated encryption ciphers, used mainly for Galois/Counter Mode (GCM) mode of Advanced Encryption Standard encryption.
- Encrypted PreMasterSecret version numbers verification.

3. OpenSSL basics

OpenSSL is a free, full-featured SSL implementation currently available for use with the C and C++ programming languages. It is an inherited work from SSLeay and is supported by UNIX and most versions of Microsoft Windows operating systems. In December 1998, development of SSLeay ceased, and the first version of OpenSSL was released as 0.9.1c, using SSLeay 0.9.1b. The OpenSSL toolkit is licensed under an Apache-style license, which basically means that you are free to get and use it for commercial and non-commercial purposes subject to some simple license conditions. OpenSSL is essentially two tools in one: a cryptography library and an SSL toolkit. There are no other SSL implementations in C which are free and available for commercial use.

4. System design

Fig. 1 shows the high level TLS 1.2 design and OpenSSL Library modules are depicted in Fig. 2. Secure communication can be achieved using TLS 1.2 protocol which is a part of SSL library. Application data is passed through the reliable transport protocol e.g. TCP through SSL library as SSL packet. SSL packet is further divided into transport layer packet and sent across to other entity.

TLS 1.2 Module: This module is used to allow peers to agree upon security parameters for the record layer, authenticate themselves, instantiate negotiated security parameters and report error conditions to each other.
TLS 1.2 API Module: This module provides the additional user interface required for TLS 1.2 related information (Supported sign-hash extensions for client hello and certificate extensions) to SSL object.
TLS1.0 module: This module provides functionality of TLS1.0.

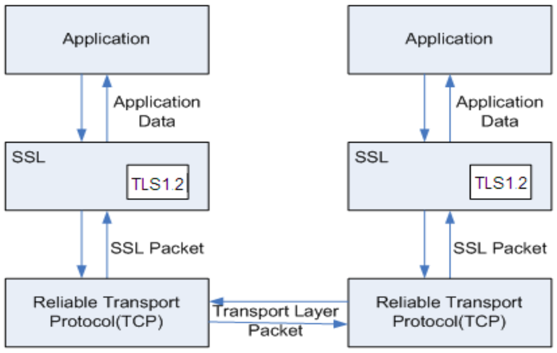

Fig 1: High level TLS 1.2 design

S2 and S3 module: These modules support functionality of SSL 2.0 and SSL 3.0 respectively.
S23 module: This module supports compatibility of SSL 2.0 and SSL 3.0.
SSL ciphersuite module: This module supports all SSL, TLS 1.0 and TLS 1.2 cipher suites.
SSL PKI handle module: This module supports Public Key Infrastructure (i.e. certificate).
SSL cryptography module: This module supports all kind of cryptographic operations.
SSL socket module: This module supports bind of socket to SSL object.
SSL error handle and log module: This module supports logging of all error messages.

The cryptographic parameters of the session state are produced by the TLS Handshake Protocol which uses messages to negotiate the cipher suite and authenticate the server to the client and to exchange information for building the cryptographic secrets.

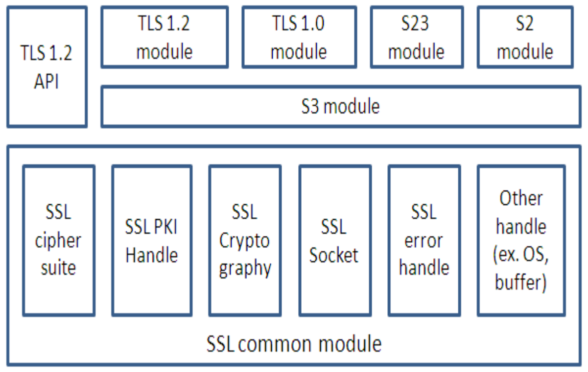

Fig 2: Modules involved in OpenSSL library for TLS 1.2

The TLS Handshaking is done in four phases:
- Establishing security capability through Client Hello and Server Hello messages.
- Authentication is achieved by exchange of keys between client and server.
- Client Key Exchange and authentication. The client is authenticated to the server. Both the client and the server know the pre-master secret.
- Finalizing and Finishing. Now the client and the server are ready to exchange data.

The high level changes required in OpenSSL library to support TLS 1.2 are:
- SHA256 is implemented in the cryptography library. For SSL corresponding ID and cipher needs to be supported.
- While using TLS1.2 the PRF function should use SHA256.
- New TLS1.2 ciphersuite based on SHA256 should be added.
- Protocol negotiation should now include TLS1.2 version checking.

Signature algorithm extension

Client uses this extension to indicate to the server which signature/hash algorithm pairs may be used in digital signatures.

1. Changes required in client for Client Hello message: Client adds the hash and signature algorithm pair that it can support as extension
2. Changes required in server for Client Hello message: Parsing of the signature algorithm extension is required. While selecting cipher suite the server must check all certificates that are signed by a hash/signature algorithm pair that appears in the extension. Extension present is detected by whether there are bytes following the compression methods at the end of the Client Hello. If client doesn't send signature algorithms use SHA1, RSA pair.
3. Changes required in Server Certificate message: Selected certificates must be signed by Signature and Hash algorithm pairs protected by client.
4. Changes required in client for Server Certificate message: Client must verify server certificate chain provided by server is signed with client supported sign-hash pair.
5. Changes in server required for Server key exchange message: Signature and hash algorithm used for singing parameter must be one of the pair provided by client. The hash and signature algorithms must be compatible with the key in the server's end-entity certificate.

Certificate type and CA check in Certificate Request

TLS 1.2 adds Certificate type and CA checks as part of certificate request processing at client side. Changes required in Client for Certificate request message: Apart from parsing certificate types, client must use certificate types for selecting the client certificate. If the certificate authorities list in the certificate request message is not empty, then one of the certificates in the certificate chain should be issued.

Signature/Hash Algorithm extension in Certificate request

TLS 1.2 adds supported_signature_algorithms as part of certificate request.

1. Changes required in Server for Certificate Request message: In addition to cert types, server adds the Hash and signature algorithm pairs that server supports in certificate request.
2. Changes required in Client for Certificate Request message: End entity certificate public key has to be compatible with certificate types listed in Certificate request. The certificates must be signed using an acceptable hash/signature algorithm pair.

3. Changes required in Client for Client Certificate: Client sends the selected Client certificate chain. If no certificate is match server criteria, it must send the client certificate message containing no certificate.
4. Changes required in Server for Client Certificate: Server verifies the Client certificate chain. During verification check whether the certificate provided by client are signed with server supported sign-hash algorithm pairs.

5. Implementation

The list below shows the OpenSSL API call flow for TLS 1.2 protocol.
1. Global system initialization
SSL_library_init();
SSL_load_error_strings()
2. Application creates a TCP socket connection
tem=accept(s,(structsockaddr*)&from,(void*)&len);
3. Get all the function ptr for TLS 1.2 implementation
meth=TLSv1_2_method();
4. Create a new global context to store all global configurations
ctx=SSL_CTX_new(meth);
5. Load certificate and private key
SSL_CTX_use_certificate_chain_file(ctx, keyfile);
SSL_CTX_use_PrivateKey_file(ctx, keyfile, SSL_FILETYPE_PEM);
6. Load the CAs we trust
SSL_CTX_load_verify_locations(ctx, CA_LIST,0);
7. Create a context per connection (SSL object)
ssl = SSL_new(ctx);
8. Create I/O abstraction for the socket
sbio = BIO_new_socket ((int)serverSocket, BIO_NOCLOSE);
9. Associate the I/O abstraction with the SSL object
SSL_set_bio(ssl,sbio,sbio);
10. Perform SSL handshake
ret = SSL_accept(ssl);
11. Write data securely
ret = SSL_write(ssl,writeBuf,writeLen);
12. Read data securely
ret = SSL_read(ssl,buf,1024);
13. Send close_notify alert to securely teardown the connection ret = SSL_shutdown(ssl);
14. Free the SSL context and close the connection
SSL_free(ssl);
closesocket(serverSocket);

Configurations:

1. SSL_METHOD *TLSv1_2_method(void) function acts like a constructor for TLS 1.2 functionality. SSL_METHOD structure is returned, which is populated with pointers to function implementing TLS 1.2 operations.
2. The function static int tls1_P_hash(const EVP_MD *md, const unsigned char *sec, int sec_len, const void *seed1, int seed1_len, const void *seed2, int seed2_len, const void *seed3, int seed3_len, const void *seed4, int seed4_len, const void *seed5, int seed5_len, unsigned char *out, int olen) should be modified to support the new hash mechanism EVP_DigestSign instead of HMAC_Update.
3. Attacks described in CBCATT [8] can be prevented by adding an explicit Initialization Vector to the function int tls1_enciv(SSL *s, int send).
4. The function int tls1_final_finish_mac(SSL *s, const char *str, int slen, unsigned char *out) should be modified for calculating the finished message. For TLS 1.2 the finished message should be generated using the hash algorithm that is used PRF function.
5. Add a function long ssl_get_algorithm2(SSL *s) to get the digest algorithm to be used.
6. Function SSL_ctxSetCipherList has to be modified to add new TLS 1.2 ciphers
7. The method function macro for TLS 1.2 IMPLEMENT_tls12_meth_func has to be added.

Handshake Message flow

1. s3_clnt.c – Addition of sign-hash extensions are done in ssl3_client_hello function. And new TLS 1.2 ciphers introduced.
2. s3_srvr.c – The sign-hash extensions are parsed in function ssl3_get_client_hello and the cipher based on the certificate is selected. (In this, check if the end entity certificate is having the matching sign-hash extensions as sent by the client).
3. s3_both.c – Function ssl3_output_cert_chain is modified to perform the sign-hash check for the CA. End entity certificate is taken care in the ssl3_get_client_hello, choose cipher itself.
4. s3_srvr.c–Function ssl3_send_certificate_request is updated to set the sign-hash pairs in the certificate request message.
5. s3_clnt.c–Infunction

ssl3_send_client_certificate the client certificate will be checked to see if it matches the sign-hash pairs.
6. s3_enc.c – Calculating hash of all the handshake messages is updated in function ssl3_finish_mac.
7. T1_enc.c – The new PRF implementation for TLS 1.2 is added.

6. Conclusion

SSL / TLS are the most widely deployed security protocol standard for providing authentication, integrity and secrecy. OpenSSL is primarily a library that is used by developers to include support for strong cryptography in their programs, but it is also a tool that provides access to much of its functionality from the command line. Computation of hash for file contents can be performed easily by using command-line tool. OpenSSL toolkit is licensed under an Apache-style license, means that you are free to get and use it for commercial and non-commercial purposes. TLS 1.2 is currently the most secure and up to date version of the standard. Achieving RFC compliance is very important for commercial products. In this paper how TLS 1.2 can be supported in OpenSSL is shown there by meeting the stringent security guidelines for commercial product.

References

[1] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
[2] National Institute of Standards and Technology, "Specification for the Advanced Encryption Standard (AES)" FIPS 197. November 26, 2001.
[3] B. Schneier. "Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd ed.", Published by John Wiley & Sons, Inc. 1996.
[4] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.
[5] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, April 2008.
[6] R. Rivest, A. Shamir, and L. M. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communications of the ACM, v. 21, n. 2, Feb 1978, pp. 120-126.
[7] NIST FIPS PUB 186-2, "Digital Signature Standard", National Institute of Standards and Technology, U.S. Department of Commerce, 2000.
[8] Moeller, B., "Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures", http://www.openssl.org/~bodo/tls-cbc.txt.

Suresh P received his ME degree from UVCE, Bangalore University, Bangalore, India. He is presently working as an Assistant Professor in Dept. of CSE, SVCE, Bangalore. His research interest includes Computer Network Security, Mobile Computing, Computer Architecture and Distributed Systems.

Pradeep K R received his M Tech degree from SJCE, VTU, Mysore, India. He is presently working as an Assistant Professor in Dept. of ISE, SVCE, Bangalore. His research interest includes Wireless Sensor Networks and Cloud Computing.

Maria Navin J R received his ME degree from UVCE, Bangalore University, Bangalore, India. He is presently working as an Assistant Professor in Dept. of ISE, SVCE, Bangalore. His research interest includes Computer Network Security and Distributed Systems.