vehere

WHITEPAPER

An Analysis of the Applicability of SSL / TLS Connection in Cyber Security and Forensics

Author: Debapriyay Mukhopadhyay
debapriyay.mukhopadhyay@vehere.com

Introduction:

Only a decade ago, SSL/TLS was only used by financial institutions and some specific organizations like public sector agencies for the log-in pages of security-conscious web sites and services. Today this has been expanded to almost all web-based services, and with the growth of unlawful activity this has quickly become the de-facto protocol for communications. With the growing use of encrypted traffic, the traditional approach of Network Forensics [7, 8] should also include SSL/TLS Forensics. It is the process of capturing information exchanged through SSL (TLS) connections and trying to visualize and extract meaningful information out of it so that it can help in some forensics analysis.

TLS/SSL provides authentication and encryption support to many important application layer protocols such as HTTP, POP3, IMAP, SMTP, etc. The symmetric key that gets established at the end of SSL/TLS connection establishment is used to encrypt the protocol application data. Data exchanged through these SSL/TLS connections can be subjected to man-in-the-middle attack at the security perimeter and can help a system or network administrator to see or analyze the application layer protocol data of HTTP, POP3, IMAP, SMTP, etc. as a plain text. But these devices which can do man-in-the-middle attack have limited capacity and with the newly rolled out TLS 1.3 protocol this has become almost impossible.

TLS / SSL protocol while establishing the connection leaks some significant information. In this white paper, we will see how we could make use of this information. Up to TLS 1.2, certificate is also sent from the server to client as a plain text and that also exposes some meaningful information. In TLS 1.3, since certificate is also sent as encrypted from server, so it is not possible to have access to this information. We will provide a TLS protocol wise detailed comparison of what could be achieved.

SSL/TLS fingerprinting is a mechanism which was introduced way back in 2008 and has significantly gained attention these days. Fingerprinting of SSL Client Hello and SSL Server Hello message can help gain significant insight about the device involved in communication – for example, OS information, Platform information, device type, etc. Fingerprinting mechanism can also help identify browsers, detect malwares, etc. Cisco Joy and JA3 are two mostly used SSL fingerprinting mechanisms. We will describe in detail these fingerprinting mechanisms and the information these fingerprints reveal.

The organization of the white paper is as follows. We will first give a closer look into the major variants of SSL/TLS protocol. Next, we will describe what information could be achieved from the SSL/TLS protocol messages. Finally, we will see the details about the SSL / TLS fingerprinting technology. Throughout this white paper will also refer different Open-Source tools and databases which one can make use of to gain insights from the SSL traffic.

SSL/TLS Protocol :

Encryption of the application layer protocol data is required to enforce security while communicating over the internet. For multiple decades SSL and its descendent, TLS, have ensured this encryption to enforce security. Secure Sockets Layer, in short SSL, is the protocol which was developed in mid-1990s by a company called Netscape. After which it was standardized by IETF and released after making some changes to the earlier one and was given a new name called Transport Layer Security or TLS. This version of the protocol, popularly known as TLS 1.0 was released in 1999 and published as RFC 2246 [1]. Next version of the protocol, known as TLS 1.1 is released in 2006 and published as RFC 4346 [2]. Tightening up various requirements to provide enhanced security, next version of the protocol TLS 1.2 came in 2008 and published as RFC 5246 [3]. Till 2019, TLS 1.2 is the dominant version of TLS and has many improvements over TLS 1.1. TLS 1.3 makes further improvements on top of TLS 1.2 and has made a paradigm shift in its design. It is the latest version of TLS protocol and has been ratified as RFC 8446 [4] in 2018.

SSL/TLS protocol includes both asymmetric and symmetric cryptography to provide security. To provide encryption/ decryption of the protocol data, both client and server is required to agree on a common shared key. This key is used to encrypt and decrypt the protocol data at transmitting and receiving end respectively and this involves symmetric key cryptography. Question is, "How do client and server agree on a common shared key"? The process through which client and server agree on a common shared key is called handshake and this handshake process mainly defines the SSL or TLS protocol. Asymmetric cryptography is used here in the handshake process to facilitate secure computation of the common shared key by both client and server.

Asymmetric cryptography is computationally expensive and that is why it is only used in the handshake process and to encrypt or decrypt the protocol traffic significantly faster symmetric cryptography is used. It is important to see the handshake process in detail with regards to TLS 1.2 and TLS 1.3.

SSL/TLS Handshake Process :

End goal of using SSL/TLS over application layer protocol is not just to be able to encrypt or decrypt protocol traffic, but also to provide solutions to problems like authentication, data integrity, and many more. Handshake process tries to address most of these problem areas and that is why it is complex and broad. To describe each and every aspect of this is beyond the scope of this White paper and we will provide only a outline of the handshake process here.

	Time	Source	Destination	Protocol	Length	Info
24	0.415298	169.254.255.66	169.254.100.98	TLSv1	168	Client Hello
25	0.415502	169.254.100.98	169.254.255.66	TLSv1	824	Server Hello, Certificate, Server Hello Done
26	0.416972	169.254.255.66	169.254.100.98	TLSv1	380	Client Key Exchange, Change Cipher Spec, Encrypted Handshake …
27	0.420775	169.254.100.98	169.254.255.66	TLSv1	113	Change Cipher Spec, Encrypted Handshake Message
28	0.422296	169.254.255.66	169.254.100.98	TLSv1	171	Application Data
29	0.422466	169.254.100.98	169.254.255.66	TLSv1	171	Application Data

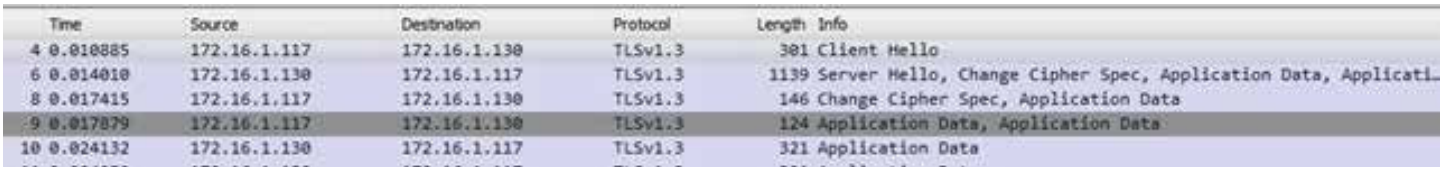
Figure 1: Basic SSL/TLS protocol flow up to TLS 1.2

SSL / TLS client starts by sending a "Client Hello" message to the SSL / TLS server by publishing its own capabilities and the different security features that it supports. Server responds back with a "Server Hello" message wherein it helps client to know the cipher suite with which both of them can continue with the negotiation. Through "Server Hello" message server also agrees to use some of the security features that client has previously published.

Server then sends its digital certificate through a message called "Server Certificate" and this certificate is issued

by some trusted authority called CA (Certificate Authority) and on having received and verifying it, client becomes sure of continuing with the connection with an authenticated server. Server certificate contains the private key of the server. Client then secretly chooses a pre-shared key and using the server's public key encrypts it and send it to server through a message called "Client Key Exchange". Only server can decrypt it and can get to know about the pre-shared key, since it can only be decrypted by the matching private key of the server. This pre-shared key is then used by both client and server to establish the common shared key with which the protocol data is encrypted and sent across.

To mark the end of the handshake process and to verify the sanity of the agreed upon shared secret, both parties send to each other a "Encrypted Handshake Message". If none of the client and server on having received the "Encrypted Handshake Message" does not respond with a SSL Alert Message, then that signifies a successful Handshake is done and both the ends are ready now to encrypt/decrypt application traffic.

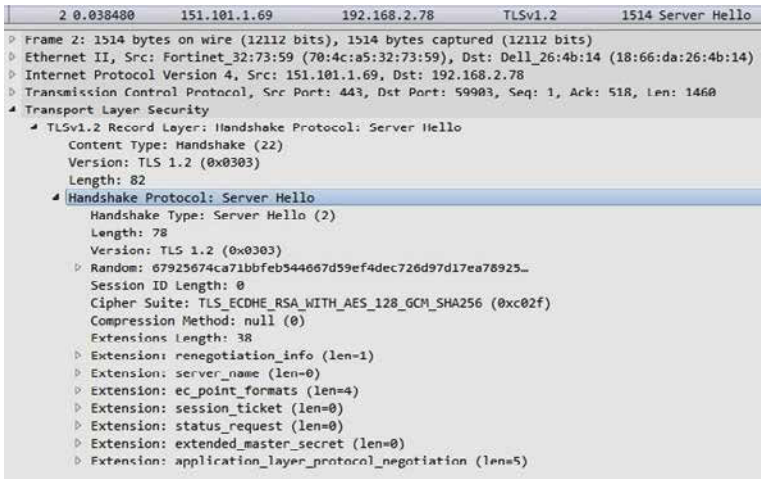


	Time	Source	Destination	Protocol	Length	Info
4	0.010885	172.16.1.117	172.16.1.130	TLSv1.3	301	Client Hello
6	0.014010	172.16.1.130	172.16.1.117	TLSv1.3	1139	Server Hello, Change Cipher Spec, Application Data, Applicati…
8	0.017415	172.16.1.117	172.16.1.130	TLSv1.3	146	Change Cipher Spec, Application Data
9	0.017879	172.16.1.117	172.16.1.130	TLSv1.3	124	Application Data, Application Data
10	0.024132	172.16.1.130	172.16.1.117	TLSv1.3	321	Application Data

Figure 2: TLS 1.3 protocol flow

This basic philosophy of SSL/TLS handshake remains same in almost all versions of TLS including the latest TLS 1.3 as well. In TLS 1.3, immediately after exchanging "Client Hello" and "Server Hello" messages, both parties agree on a secret key which is referred as handshake key. And this handshake key is in use to encrypt/decrypt the other messages related to handshake only. So, unlike other SSL or TLS versions "Server Certificate", "Encrypted Handshake Message" are also exchanged between client and server, but now they are encrypted with handshake key. At the end, in TLS 1.3 also, both parties securely agree on a common shared secret to continue encrypting/decrypting application traffic.

Next, we will take a closer look into the relevant aspects of "Client Hello" and "Server Hello" messages because these two message are the ones which are mostly used in network forensics. "Client Hello" [Figure 4a and Figure 4b] message include a random number (called client random), a set of cipher suites published by the client (cipher suites encode capabilities of a client with which client can do a successful SSL/TLS transaction) and a set of extensions. "Server Hello" [Figure 3] message include a random number (called server random), a single cipher suite chosen from the list published by client with which server wants the SSL handshake negotiation and transaction to get completed and a set of extensions. There are other fields, though are present, but that is outside the scope of discussion for the current white paper.



```
2 0.038480   151.101.1.69   192.168.2.78   TLSv1.2   1514 Server Hello
▷ Frame 2: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
▷ Ethernet II, Src: Fortinet_32:73:59 (70:4c:a5:32:73:59), Dst: Dell_26:4b:14 (18:66:da:26:4b:14)
▷ Internet Protocol Version 4, Src: 151.101.1.69, Dst: 192.168.2.78
▷ Transmission Control Protocol, Src Port: 443, Dst Port: 59903, Seq: 1, Ack: 518, Len: 1460
◢ Transport Layer Security
  ◢ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 82
    ◢ Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 78
        Version: TLS 1.2 (0x0303)
      ▷ Random: 67925674ca71bbfeb544667d59ef4dec726d97d17ea78925…
        Session ID Length: 0
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
        Compression Method: null (0)
        Extensions Length: 38
      ▷ Extension: renegotiation_info (len=1)
      ▷ Extension: server_name (len=0)
      ▷ Extension: ec_point_formats (len=4)
      ▷ Extension: session_ticket (len=0)
      ▷ Extension: status_request (len=0)
      ▷ Extension: extended_master_secret (len=0)
      ▷ Extension: application_layer_protocol_negotiation (len=5)
```

Figure 3: Server Hello packet

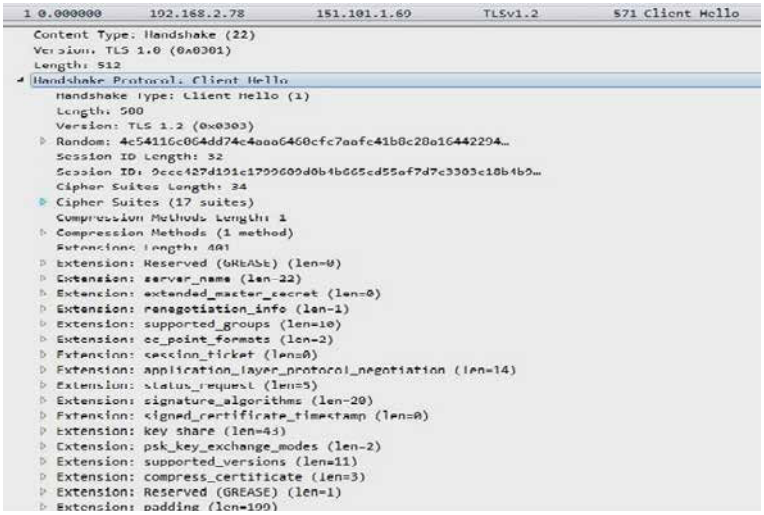


Figure 4a: Client Hello Packet

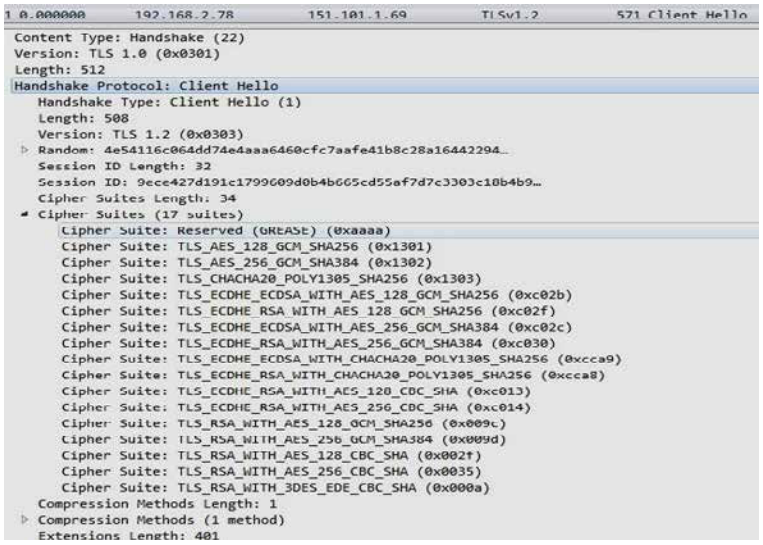


Figure 4b : Client Hello packet showing
all published cipher suites

Insights achieved from SSL/TLS protocol :

The major sources of gaining information out of SSL/TLS protocol are “Client Hello” and “Server Hello” messages. In this Section, we will see systematically how we can gain insights from these two messages and we start by defining Cipher Suite which is central to any SSL/TLS protocol. Cipher Suite is a collection of algorithms which are used in any SSL/TLS protocol to provide solutions to different security problems (authentication, data integrity, encryption, or decryption, etc.). Each cipher suite has a unique name, and it is defined by the SSL/TLS protocol.

For example, *TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256*, where TLS depicts the protocol, it is for, ECDHE (elliptic curve Diffie-Hellman algorithm) is to be used for key generation, RSA will be used for authenticating the server, the block cipher algorithm AES_128_GCM will be used to encrypt or decrypt protocol data and lastly SHA256 is used to ensure data integrity. The key generation algorithm which in this case is ECDHE, has a ‘E’ at the end which means the generated key will be ephemeral. There are other variants of key generation algorithm such as ECDH (generated key is not ephemeral), DHE, DH and RSA. DHE and DH are non-elliptic versions of Diffie-Hellman algorithm. When RSA is used, it is used for both key generation and server authentication.

Forward Secrecy or Perfect Forward Secrecy is a security feature demanded from the SSL/TLS protocol to make sure that even if the public-private key pair of the server is compromised, it should not enable an adversary to break all past communications. This requires, separate session key (or shared key) is to be generated for each session and when that is done it is called an Ephemeral key. So, obviously ephemeral versions of key generation are more secure than the non-ephemeral versions such as ECDH, DH and RSA. Therefore, it is an interesting insight to check whether ephemeral cipher suite has been used or not in establishing the SSL/TLS connection. In TLS 1.3, the cipher suites used are all ephemeral. So, when we see that in response to client's request (to establish a TLS connection with TLS 1.3), server has also agreed to proceed with the same version, then we can be assured of using an ephemeral key generation algorithm.

Figure 5: Showing contents of some important extensions in Client Hello

There is a mandatory extension server_name (Extension type equals to 0) in "Client Hello" packet which tells us the name of the server with which client is trying to establish a SSL/TLS connection. Note, in [Figure 5] client has initiated a connection to the server stackoverflow.com.

In [Figure 5], immediately after the 'server_name' extension, the extension that shows up is 'extended_master_secret' (Extension type equals to 23). and the use of this extension signifies creation of shared (master) secret in a way which guarantees that no two SSL connections will have the same secret when there is a proxy server sitting in between the client and server.

The use of 'extended_master_secret' happens when in response to client's request, in 'Server Hello' message server also appends the same extension. If server does not send this extension, in its 'Server Hello' message, then SSL connection will land up creating the shared secret in non-extended way. For advanced reader, one can refer [11] to know the difference between these two ways of creating the shared or master secret.

Next, message that follows 'Server Hello' is 'Server Certificate' from the server end. This message though is visible in plain text up to TLS 1.2 and in TLS 1.3 this is encrypted and will not be of any help in drawing some insights out of it. This certificate reveals many information such as identity of the certificate issuer, version, serial number, algorithm details, issuer name, validity period, and other significant details of PKI. There is a field called RDN sequence in the certificate from which we can get to know about many important details, and they remain in the certificate as attribute value pair. These attributes that can be part of RDN sequence are common name, surname, serial number,

country name, locality name, street or province name, street address, organization name, organizational unit, title, email address, user id, domain component. Certificate also contains a few extensions. One important extension is subject alternative name which allows us to know the additional host name which are also protected by the same certificate for mainly multi-domain cases.

The use of un-trusted digital certificates [9] are growing as malware authors are now relying on SSL connection to sneak past Intrusion Detection and any protection system. Switzerland's Abuse.ch has created a repository which keeps track of blacklisted certificates that have been associated with banking malware, malware campaign and botnets. And this has been created to help security and digital forensic professionals to figure out Blacklisted SSL certificate. In this site https://sslbl.abuse.ch/blacklist/, they have provided a freely downloadable CSV containing the SHA1 fingerprint of all the blacklisted certificates found so far.

SSL/TLS Fingerprint :

We will now describe essential details of SSL/TLS fingerprinting technique. Since SSL/TLS handshake process includes sending 'Client Hello' and 'Server Hello' packet in plain text, so the method to identify the browser, or the application or the OS of the system is known as SSL/TLS fingerprinting. And this method predominantly uses the details used in 'Client Hello' or 'Server Hello' packets. The fingerprint generated out of 'Client Hello' packet is called client-side SSL/TLS fingerprint, and if it is formed out of 'Server Hello' packet, then it is called server-side fingerprint.

The JA3 method [5] of forming SSL fingerprint extracts the decimal value of the bytes of certain fields from the 'Client Hello' packet. It considers the following fields such as SSL/TLS version, list of published cipher suites, list of extensions, list of elliptic curves and list of elliptic curve point formats. All these decimal values are then concatenated in the same order as has been mentioned above. While concatenating, it delimits each field by a ',' and each value in each field by a '-'. The string thus formed, is then MD5 hashed, and this hash value is considered as client side SSL / TLS fingerprint.

For example, for the Client Hello in Figure [4a], the concatenated string extracting different SSL / TLS features will look like the following.

771,4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53-10,0-23-65281-10-11-35-16-5-13-18-51-45-43-27-21,29-23-24,0

The MD5 hash of this string *66918128f1b9b03303d77c6f2eefd128* is considered as the client-side SSL/TLS fingerprint. If there are no TLS extension in Client Hello, or extension like supported groups (extension type 10) are missing, then the fields are left empty. One point to make here is that GREASE values are often part of cipher suites and extensions. While forming the string for SSL fingerprint generation, JA3 method ignores the GREASE values. Also.

To extend the same method to fingerprint the server side of the SSL / TLS handshake using the 'Server Hello' message we use the method JA3S. The SSL/TL features which are in consideration from 'Server Hello' message are version, accepted cipher, list of extensions. The string is formed in the same was as that of JA3 following specifically the order mentioned. For example, for the Server Hello in Figure [3], the concatenated string extracting different SSL / TLS features will look like the following.

771,49199,65281-0-11-35-5-23-16

The MD5 hash of this string *860fcf58fd757e26aa8911e5eaff6b53* is considered as the server side SSL/TLS fingerprint.

Many servers and clients use different SSL/TLS features and thus giving us the scope to identify them with respect to their SSL/TLS fingerprints. Custom software to inject malware over SSL, uses specific set of SSL/TLS features (at times in different orders other than what has been used normally by browsers, or normal SSL/TLS stack used in

securing the network) and in turn it facilitates its detection (the communication with malware software) through SSL/TLS fingerprint as it will be unique. In certain cases, combination of JA3 + JA3S fingerprints will help in isolating anomalous connection over SSL/TLS.

BSD licensed software package Joy of Cisco [6] also provides SSL/TLS fingerprinting methodology by extracting SSL/TLS features from 'Client Hello' packet. Like JA3, this fingerprint also works for all versions of SSL/TLS. Following SSL / TLS features are extracted from 'Client Hello' packet.
TLS version, list of all cipher suites including the grease value, and all extension types including the grease value are used. For a few extensions, the whole content of the extension is used and those extensions are: supported_groups, ec_point_formats, signature_algorithms, supported_versions, status_request, application_layer_protocol_negotiation and psk_key_exchange_versions.

The main difference with JA3 here is that in JA3 the whole content of the extensions mentioned above are not in use. Also, the way string is formed is different. JA3 considers the decimal value while forming the string, whereas in Cisco Joy hexa-decimal value has been considered. Like JA3, string thus formed is MD5 hashed to produce the SSL/TLS fingerprint. For the 'Client Hello' in Figure [4a], the concatenated string as per Cisco Joy's fingerprinting mechanism will look like the following.

(0303)(aaaa130113021303c02bc02fc02cc030cca9cca8c013c014009c009d002f0035000a)((2a2a)(0000)(0017)(ff01)(000a000a0008eaea001d00170018)(000b00020100)(0023)(0010000e000c02683208687474702f312e31)(00050005010000000000)(000d00140012040308040401050308050501080606010201)(0012)(0033)(002d00020101)(002b000b0a8a8a0304030303020301)(001b)(8a8a)(0015))

The MD5 hash of this string *2fe06655aa385fa426b82666cc7331c0* is considered as the client side SSL/TLS fingerprint. Cisco Joy also provides a json file, where each entry in the file refers to different ways of using SSL / TLS features (from 'Client Hello') by different applications or operating systems and thereby helping us to detect with some probability the application, process, browser or operating system [10] of the computing device in use. This is a huge set of information for use by any network forensic professional.

Conclusion

With the growth of cyber assisted crime, there is an increasing importance on the incident management capabilities to detect and prevent misuse of systems. Network forensics has got huge success in providing that insight to pin point the trail of any malicious or un-lawful activity. While SSL/TLS in one hand is trying to secure our network, on the other hand its use by adversaries have added complexity in extracting meaningful information from encrypted network traffic. Therefore, trying to extract information from SSL / TLS protocol, can help in gaining insight by coupling it with information extracted from other plain text protocols to achieve the desired goal of network forensics. We have kept our attention focussed entirely on SSL/TLS protocol for analysis and the method of decrypting the application traffic by an interceptor in the middle for forensics analysis has been kept outside the scope of this white paper.

References

- https://www.ietf.org/rfc/rfc2246.txt
- https://www.ietf.org/rfc/rfc4346.txt
- https://www.ietf.org/rfc/rfc5246.txt
- https://www.ietf.org/rfc/rfc8446.txt

- https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967
- https://github.com/cisco/joy
- G. Shrivastava, "Network forensics: Methodical literature review," 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, 2016, pp. 2203-2208.
- Suleman Khan, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Muhammad Shiraz, Iftikhar Ahmad, "Network forensics: Review, taxonomy, and open challenges,"Journal of Network and Computer Applications, Volume 66, 2016, Pages 214-235, ISSN 1084-8045.
- Soghoian C., Stamm S. (2012) "Certified Lies: Detecting and Defeating Government Interception Attacks against SSL (Short Paper)", In: Danezis G. (eds) Financial Cryptography and Data Security. FC 2011. Lecture Notes in Computer Science, vol 7035.
- M. Husák, M. Cermák, T. Jirsík and P. Celeda, "Network-Based HTTPS Client Identification Using SSL/TLS Fingerprinting," 2015 10th International Conference on Availability, Reliability and Security, Toulouse, 2015, pp. 389-396, doi: 10.1109/ARES.2015.35.
- https://datatracker.ietf.org/doc/html/rfc7627