MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
A Constituent Institution of Manipal University

Department of Computer Science & Engineering

Capture the TLS handshaking and figure out the types of cipher suits in use in all the sessions

A MINI PROJECT REPORT

Submitted by:

Harshika Sofat (200905117)
Anushka Das (200905163)

In partial fulfillment for the award of the degree of

B.Tech in Computer Science Engineering In

Computer Network Labs

Department of Computer Science & Engineering

BONAFIDE CERTIFICATE

Certified that this project report "Capture the TLS handshaking and figure out the types of Cipher suits in use in all the sessions" is the bonafide work of

Harshika Sofat (200905117)
Anushka Das (200905163)

who carried out the mini project work under my supervision.

Dr. Ashalatha Nayak	Mr. RadhaKrishna Bhat
Professor and HOD	Professor, CSE

Submitted to the Viva voce Examination held on 2nd of November, 2022.

EXAMINER 1	EXAMINER 2

ACKNOWLEDGEMENT

We want to express our most significant appreciation to all the individuals who have helped and supported us throughout the project. We are thankful to our COMPUTER NETWORKS LAB teachers for their ongoing support during the project, from initial advice and encouragement, which led to the final report of this project. I would also like to thank

Mr. Radhakrishna Bhat and **Ms. Archana Praveen Kumar** who always guided us in the computer lab. A special acknowledgment goes to our classmates who helped us complete the project by exchanging interesting ideas and sharing their experiences.

We wish to thank our parents for their undivided support and interest in our interests and well-being, inspiring us and encouraging us to go our own way. Without their support and blessing, we wouldn't be in this position.

In the end, we want to thank our friends who displayed appreciation for our work and motivated us to continue our work.

INDEX

ABSTRACT

The problem given in hand was to capture the packets of TLS handshaking and researching on types of cipher suites, which we were able to implement using wireshark and terminal. We have attached screenshots of all the steps and given a detailed explanation about each of them. TLS is a cryptographic protocol that provides end-to-end security of data sent between applications over the Internet. Cipher suites are sets of instructions that enable secure network connections through Transport Layer Security (TLS), often still referred to as Secure Sockets Layer (SSL).

INTRODUCTION TO TLS

WHAT IS TLS?

Transport Layer Security (TLS) encrypts data sent over the Internet to ensure that eavesdroppers and hackers are unable to see what you transmit which is particularly useful for private and sensitive information such as passwords, credit card numbers, and personal correspondence.

TLS is a cryptographic protocol that provides end-to-end security of data sent between applications over the Internet. It is mostly familiar to users through its use in secure web browsing, and in particular the padlock icon that appears in web browsers when a secure session is established. However, it can and indeed should also be used for other applications such as e-mail, file transfers, video/audioconferencing, instant messaging and voice-over-IP, as well as Internet services such as DNS and NTP.

The TLS protocol ensures privacy and data security by encryption so that any third party cannot intercept the communication; it also authenticates the peers to verify their identity. By providing a secure channel of communication between two peers, TLS protocol protects the integrity of the message and ensures that it is not being tampered with.

TLS evolved from Secure Socket Layers (SSL), which was originally developed by Netscape in 1994, to secure web sessions. SSL 1.0 was never publicly released, whilst SSL 2.0 was quickly replaced by SSL 3.0, on which TLS is based. However, SSL 3.0 is now considered insecure and was deprecated by RFC 7568 in June 2015, with the recommendation that TLS 1.2 and 1.3 should be used.

WHY IS TLS REQUIRED?

Without TLS, sensitive information such as logins, credit card details and personal details can easily be gleaned by others, but also browsing habits, e-mail correspondence, online chats and conference calls can be monitored. By enabling client and server applications to support TLS, it ensures that data transmitted between them is encrypted with secure algorithms and not viewable by third parties. Recent versions of all major web browsers currently support TLS, and it is increasingly common for web servers to support TLS by default.

It is therefore recommended that all clients and servers insist on mandatory usage of TLS in their communications, and preferably the most recent version TLS 1.3. For complete security, it is necessary to use it in conjunction with a publicly trusted X.509 Public Key Infrastructure (PKI) and preferably DNSSEC as well in order to authenticate that a system to which a connection is being made is indeed what it claims to be.

HOW DOES TLS WORK ?

TLS uses a combination of symmetric and asymmetric cryptography, as this provides a good compromise between performance and security when transmitting data securely.

With symmetric cryptography, data is encrypted and decrypted with a secret key known to both sender and recipient; typically 128 but preferably 256 bits in length.

Asymmetric cryptography uses key pairs – a public key, and a private key. The public key is mathematically related to the private key, but given sufficient key length, it is computationally impractical to derive the private key from the public key. This allows the public key of the recipient to be used by the sender to encrypt the data they wish to send to them, but that data can only be decrypted with the private key of the recipient. TLS uses asymmetric cryptography for securely generating and exchanging a session key. The session key is then used for encrypting the data transmitted by one party, and for decrypting the data received at the other end. Once the session is over, the session key is discarded.

With TLS it is also desirable that a client connecting to a server is able to validate ownership of the server's public key. This is normally undertaken using an X.509 digital certificate issued by a trusted third party known as a Certificate Authority (CA) which asserts the authenticity of the public key.

WHAT IS A CA?

A Certificate Authority (CA) is an entity that issues digital certificates conforming to the ITU-T's X.509 standard for Public Key Infrastructures (PKIs). Digital certificates certify the public key of the owner of the certificate (known as the subject), and that the owner controls the domain being secured by the certificate. A CA therefore acts as a trusted third party that gives clients (known as relying parties) assurance they are connecting to a server operated by a validated entity.

WHEN DOES A TLS HANDSHAKE OCCUR ?

During a TLS handshake, the two communicating sides exchange messages to acknowledge each other, verify each other, establish the cryptographic algorithms they will use, and agree on session keys. TLS handshakes are a foundational part of how HTTPS works.

A TLS handshake takes place whenever a user navigates to a website over HTTPS and the browser first begins to query the website's origin server. A TLS handshake also happens whenever any other communications use HTTPS, including API calls and DNS over HTTPS queries.TLS handshakes occur after a TCP connection has been opened via a TCP handshake.

WHAT HAPPENS DURING A TLS HANDSHAKE?

During the course of a TLS handshake, the client and server together will do the following:

- Specify which version of TLS (TLS 1.0, 1.2, 1.3, etc.) they will use
- Decide on which joint cipher suite they will use
- Authenticate the identity of the server via the server's public key and the SSL certificate authority's digital signature
- Generate session keys to use symmetric encryption after the handshake is complete

PROBLEM DEFINITION AND OBJECTIVE

PROBLEM DEFINITION :

Capture the TLS handshaking and figure out the types of cipher suits in use in all the sessions

OBJECTIVE :

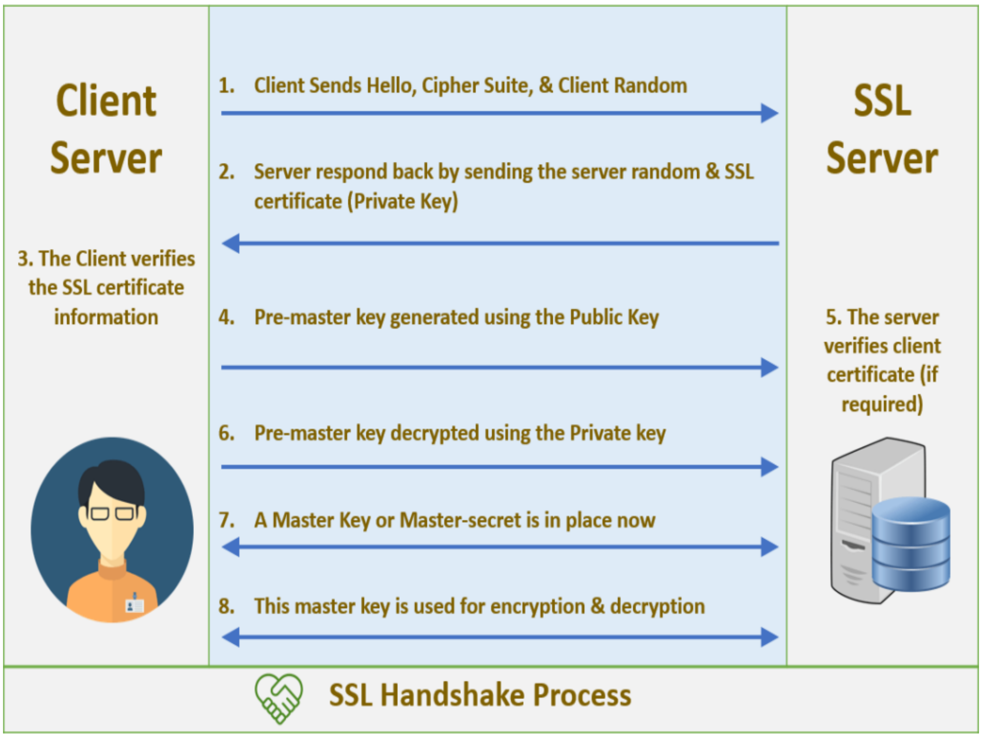
1. Aim to understand the working and implementation of TLS Handshake.
2. To learn about different cipher suites and their use in encryption of data.

IMPLEMENTATION- STEPS IN TLS HANDSHAKE

BASIC STEPS IN A TLS HANDSHAKE

The TLS handshake protocol involves the following:

1. Client initiates the handshake by sending a “client hello” message to the server. The message includes the client’s random value, supported TLS version, and cipher suite.
2. Server responds with a “server hello” message. The message includes the server’s random value, SSL certificate, chosen cipher suite, and sometimes, request of client’s certificate.
3. The client authenticates the server’s SSL certificate and the Certificate Authority.
4. The client sends the pre-master secret – a random string of bytes and encrypts it with the public key from the server's SSL certificate.
5. Server decrypts the pre-master secret, and the two parties generate the session keys.
6. Client sends a “client finished” message, encrypted with the session key.
7. Server switches its record layer security state to symmetric encryption using the session keys and sends a "Server finished" message to the client.
8. A secured channel is established, and all messages sent are encrypted using the session key.



What is a TLS handshake exception?

In some instances, the client and the server cannot agree on the desired level of security, therefore failing to establish secure communication using the TLS protocol. There are various causes for this error that will further be discussed below.

TLS handshake exceptions often cause the client to receive an HTTP status 503 with a message “service unavailable,” or “your connection isn’t private.” A 503 error means that there was a problem with the server. Some of its causes include website maintenance, error in the server’s code, or a surge in website traffic.

In SSL handshake failures, Error 525 (SSL Handshake Failed) is displayed, which indicates that the server and browser failed to establish a secure connection.

How to resolve TLS handshake exception or TLS handshake fail?

Like SSL handshake exceptions, TLS handshake fails can occur due to server-side or client-side issues. This segment will discuss the common causes of TLS handshake failures and how to resolve them.

Client-side errors:

- **Browser error**. Browser settings or plug-ins within the browser can cause errors when visiting legitimate servers. A simple solution to this problem would be changing the browser. If TLS still fails after switching browsers, check the plug-ins installed. Remove these plug-ins and restore the browser settings to default.
- **Middleman errors**. Network firewalls or other programs and devices can potentially cause handshake fails due to blocked connections. In these instances, check the devices or programs and adjust the settings.

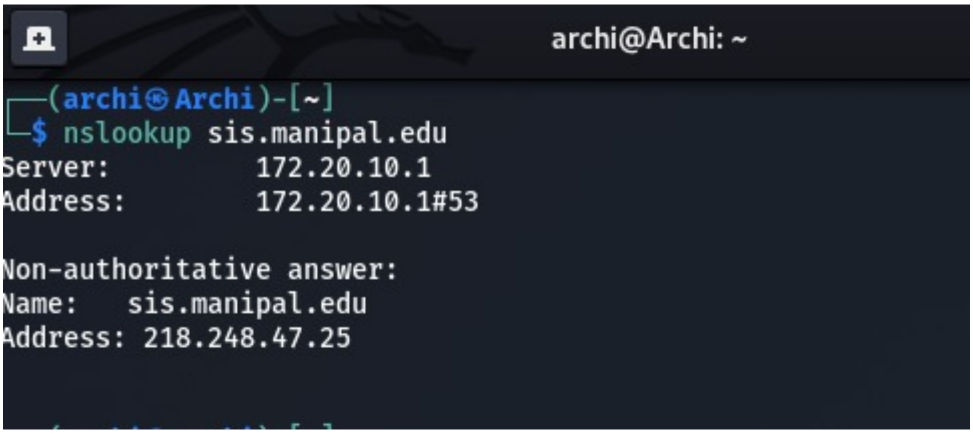
Server-side errors:

- **Protocol mismatch**. In instances where the protocol is not supported by the server, it is recommended to upgrade to the new version than going back to the old version. For instance, when the client has TLS 1.1 and the server supports 1.2, the client is recommended to upgrade to 1.2.
- **Encryption mismatch**. An error will occur when the encryption of the client is not supported by the server. To solve this, clients or servers are recommended to upgrade. Additionally, organizations must support different encryption standards.
- **Certificate error**. When certificates are non-authenticated or illegal, TLS handshakes will fail. To resolve this, the server must check their domain, certificate expiration, certificate authority legitimacy, and other details.

METHODOLOGY & OUTPUTS - TLS Handshake Captured in Wireshark

> nslookup sis.manipal.edu

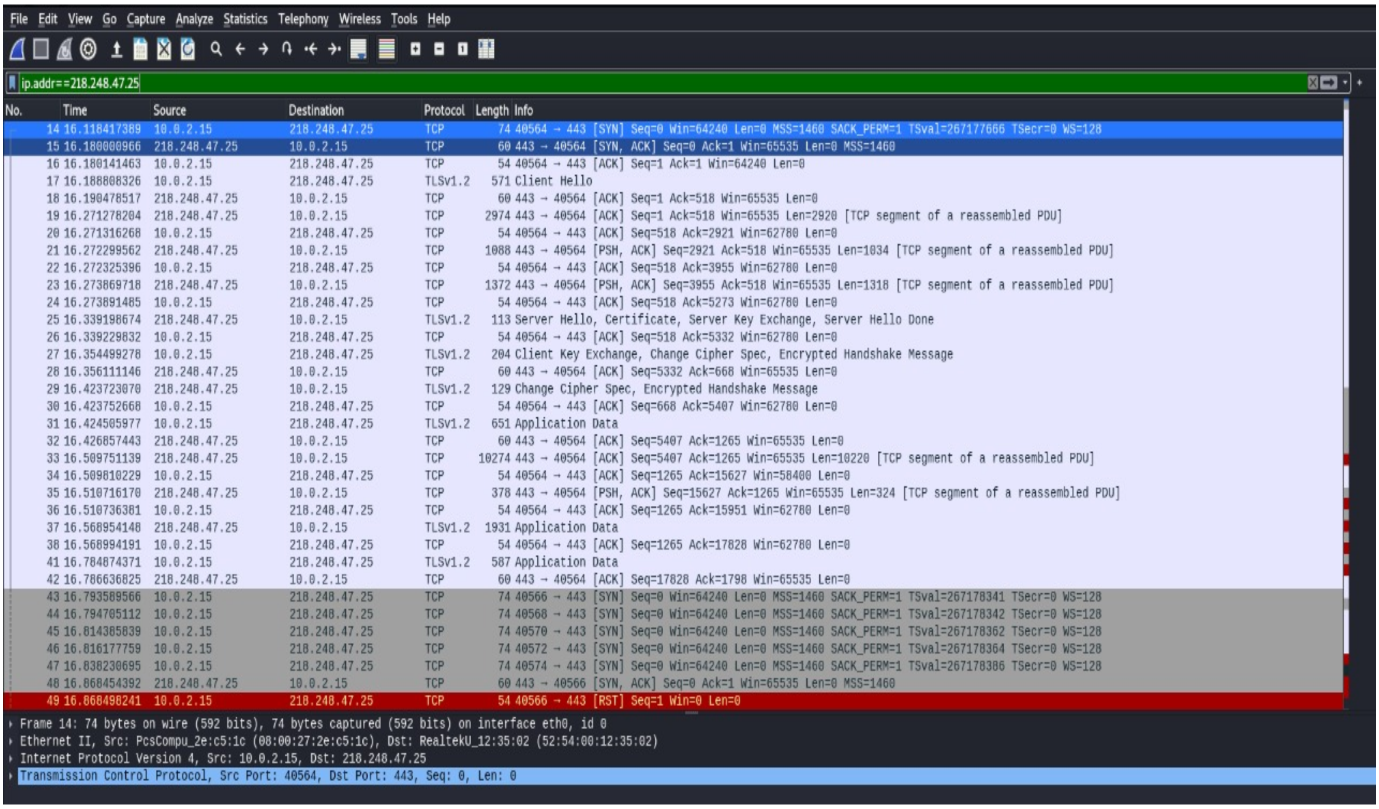
Use the *nslookup* command to look up the non-authoritative IP addresses for the DNS servers of this domain. This will be used to filter the packets in wireshark.



nslookup

We open the website and start capturing the packets on Wireshark.

*Apply the filter **ip.addr == 218.248.47.25**(obtained in nslookup step)*



Wireshark packet list with applied filter

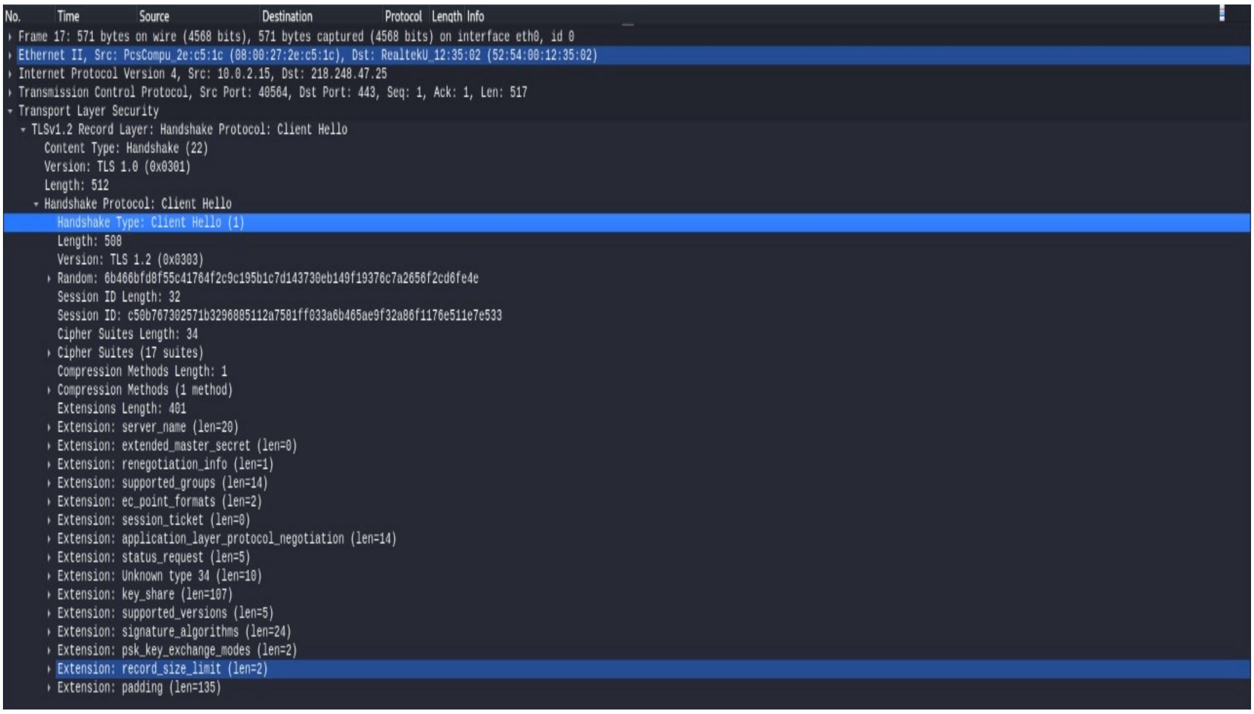
Notice that before TLS Handshake packets, the TCP handshake packets SYN, SYN, ACK and ACK are listed because an open TCP connection is necessary before TLS handshake.

Step1. Client Hello

Typically, the first message in the TLS Handshake is the client hello message which is sent by the client to initiate a session with the server.

The message contains:

- *Version*: The TLS protocol version number that the client wants to use for communication with the server. This is the highest version supported by the client.
- *Client Random*: A 32-byte pseudorandom string that is used to calculate the Master secret (used in the creation of the encryption key).
- *Session Identifier*: A unique number used by the client to identify a session.
- *Cipher Suites*: The list of cipher suites supported by the client ordered by the client's preference. The cipher suite consists of a key exchange algorithm, bulk encryption algorithm, MAC algorithm and a pseudorandom function. More about this will be discussed in chapter 3.
- *Compression Method*: Contains a list of compression algorithms ordered by the client's preference. This is optional.



<u>ClientHello</u>

```
    Cipher Suites Length: 34
  ▾ Cipher Suites (17 suites)
      Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
      Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
      Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
      Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
      Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
      Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
      Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
      Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
```

List of cipher suits sent by client to server in ClientHello

```
    Extensions Length: 401
  ▾ Extension: server_name (len=20)
      Type: server_name (0)
      Length: 20
    ▾ Server Name Indication extension
        Server Name list length: 18
        Server Name Type: host_name (0)
        Server Name length: 15
        Server Name: sis.manipal.edu
```

Server Name in the Client Hello

Step2. Server Hello

The Server Hello responds to ClientHello and contains the following information:

● *Server Version*: The highest TLS protocol version supported by the server which is also supported by the client.

● *Server Random*: 32-byte pseudorandom number used to generate the Master Secret.

● *Session Identifier*: Unique number to identify the session for the corresponding connection with the client. If the session ID in the client hello message is not empty, the server will find a match in the session cache. If a match is found and the server wants to use the same session state, it returns the same ID as sent by the client. If the server doesn't want to resume the same session, then a new ID is generated. The server can also send an empty ID, indicating the session cannot be resumed.

● *Cipher Suite*: The single strongest cipher suite that both the server and the client support. If there is no supporting cipher suite, then a handshake failure alert is created.

● *Compression Method*: The compression algorithm agreed by both the server and the client. This is optional.

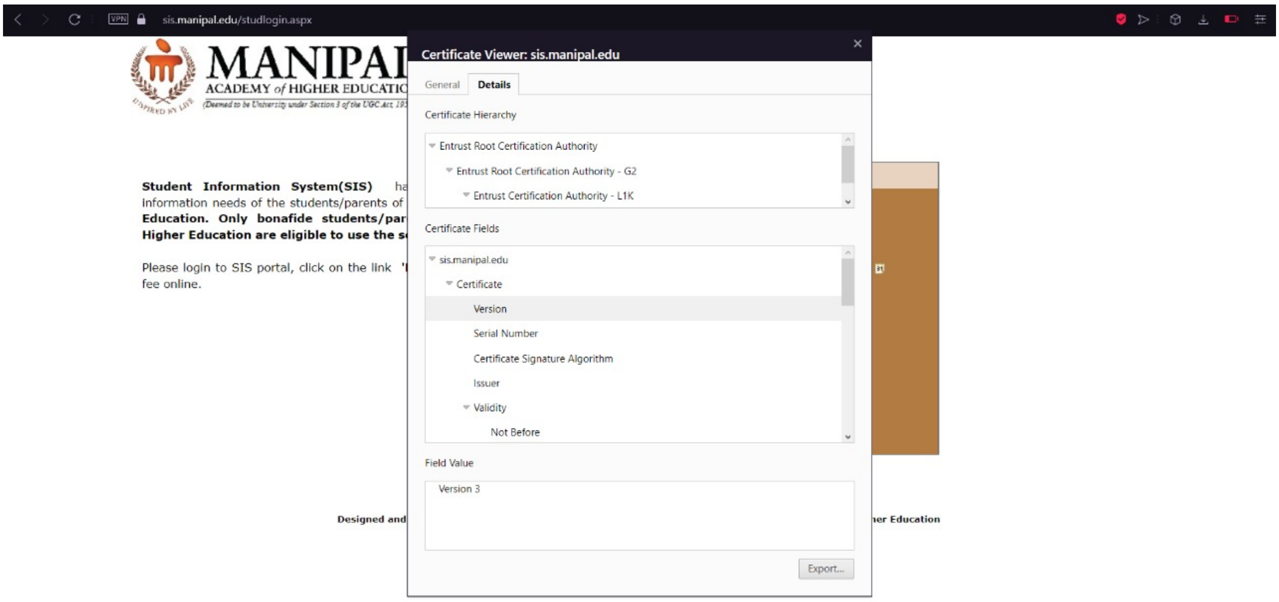
```
Transport Layer Security
  TLSv1.2 Record Layer: Handshake Protocol: Multiple Handshake Messages
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 5326
    Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 85
      Version: TLS 1.2 (0x0303)
      Random: 63602792d15b064af39c2a0ba9c8ab798089556da4bb5d8106d091e5f251ceaa
      Session ID Length: 32
      Session ID: d508000089200b23a27bbff84d6ee5ed372c68bd23de09e9e4df0bea00116312
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
      Compression Method: null (0)
      Extensions Length: 13
      Extension: extended_master_secret (len=0)
        Type: extended_master_secret (23)
        Length: 0
      Extension: renegotiation_info (len=1)
        Type: renegotiation_info (65281)
        Length: 1
        Renegotiation Info extension
      Extension: server_name (len=0)
        Type: server_name (0)
        Length: 0
```

ServerHello

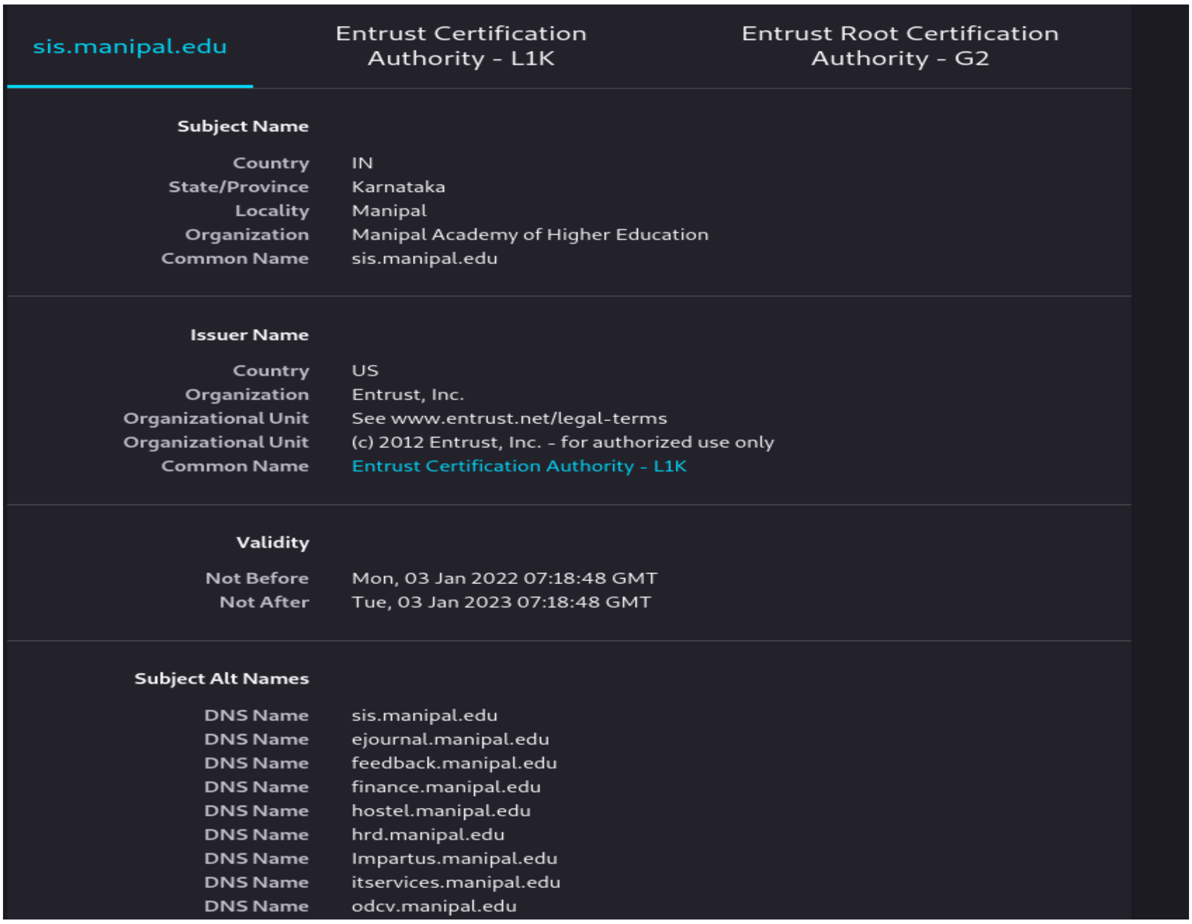
Step3. Certificate

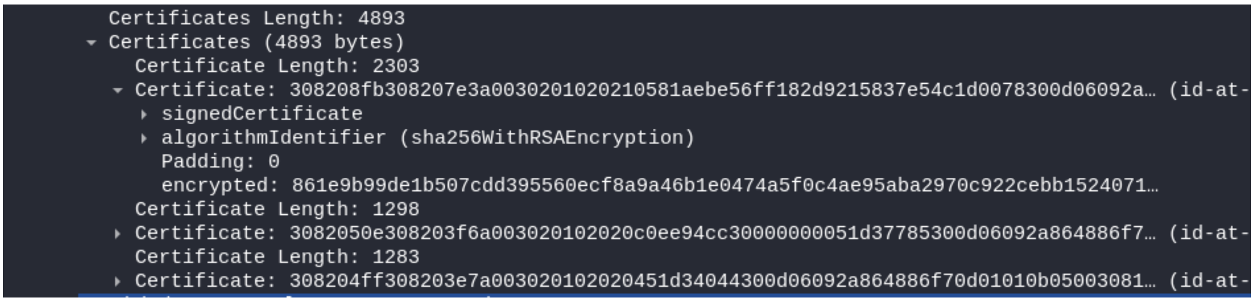
After the serverHello, server also sends the client a list of X.509 certificates to authenticate itself. The server's certificate contains its public key. This certificate authentication is either done by a third party (Certificate Authority) that is trusted by the peers, the operating system and the browser which contains the list of well-known Certificate Authorities or by manually importing certificates that the user trusts as stated earlier.

For example, the certificate for slcm.manipal.edu and its hosted aws servers are also present.



Certificates





The information sent about the certificate *.manipal.edu
(i.e.. any subdomain of manipal.edu will have same certificate as manipal.edu)

Step 4. Change Cipher Spec

This message notifies the server that all the future messages will be encrypted using the algorithm and keys that were just negotiated and mutually agreed to by both the client and the server.

```
27 16.354499278  10.0.2.15       218.248.47.25   TLSv1.2   204 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
28 16.356111146  218.248.47.25   10.0.2.15       TCP        60 443 → 40564 [ACK] Seq=5332 Ack=668 Win=65535 Len=0
29 16.423723070  218.248.47.25   10.0.2.15       TLSv1.2   129 Change Cipher Spec, Encrypted Handshake Message
30 16.423752668  10.0.2.15       218.248.47.25   TCP        54 40564 → 443 [ACK] Seq=668 Ack=5407 Win=62780 Len=0
31 16.424505977  10.0.2.15       218.248.47.25   TLSv1.2   651 Application Data
32 16.426857443  218.248.47.25   10.0.2.15       TCP        60 443 → 40564 [ACK] Seq=5407 Ack=1265 Win=65535 Len=0
33 16.509751139  218.248.47.25   10.0.2.15       TCP     10274 443 → 40564 [ACK] Seq=5407 Ack=1265 Win=65535 Len=10220 [TCP segment of a reassembled PDU]
34 16.509810229  10.0.2.15       218.248.47.25   TCP        54 40564 → 443 [ACK] Seq=1265 Ack=15627 Win=58400 Len=0

› Frame 29: 129 bytes on wire (1032 bits), 129 bytes captured (1032 bits) on interface eth0, id 0
› Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_2e:c5:1c (08:00:27:2e:c5:1c)
› Internet Protocol Version 4, Src: 218.248.47.25, Dst: 10.0.2.15
› Transmission Control Protocol, Src Port: 443, Dst Port: 40564, Seq: 5332, Ack: 668, Len: 75
▾ Transport Layer Security
  ▾ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.2 (0x0303)
      Length: 1
      Change Cipher Spec Message
```

Change Cipher Spec from Client

.

Step 5. Encrypted Handshake Message (Finished)

The Finished message is complicated as it is a hash of all the messages exchanged previously along with a label ("client finished"). This message indicates that the TLS negotiation is completed for the client. Wireshark displays the Finished message as Encrypted Handshake since, unlike the previous messages, this message has been encrypted with the just negotiated keys/algorithms.

```
▾ TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 64
    Handshake Protocol: Encrypted Handshake Message
```

Client Finished message encrypted

Step 6. Change Cipher Spec

The server informs the client that it the messages will be encrypted with the existing algorithms and keys. The record layer now changes its state to use the symmetric key encryption. (Both the client and server now use same algo to encrypt their data)

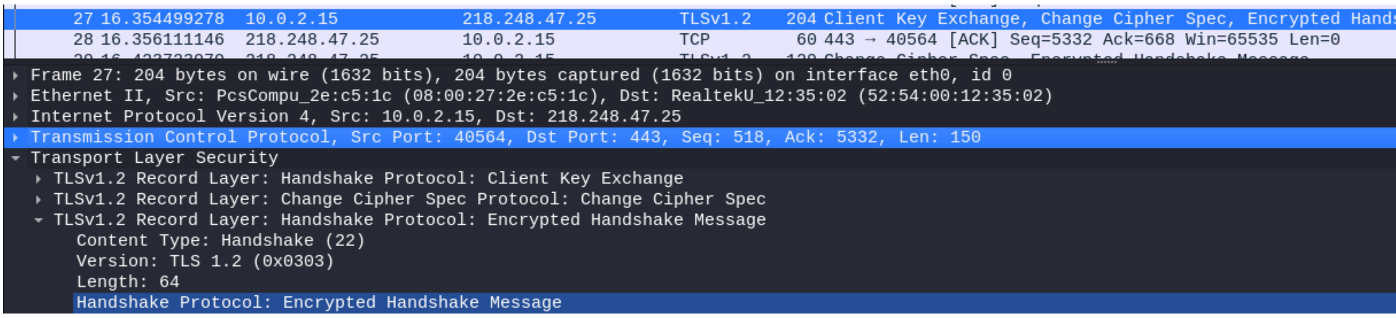
```
27 16.354499278  10.0.2.15       218.248.47.25   TLSv1.2   204 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
28 16.356111146  218.248.47.25   10.0.2.15       TCP        60 443 → 40564 [ACK] Seq=5332 Ack=668 Win=65535 Len=0
29 16.423723070  218.248.47.25   10.0.2.15       TLSv1.2   129 Change Cipher Spec, Encrypted Handshake Message
30 16.423752668  10.0.2.15       218.248.47.25   TCP        54 40564 → 443 [ACK] Seq=668 Ack=5407 Win=62780 Len=0
31 16.424505977  10.0.2.15       218.248.47.25   TLSv1.2   651 Application Data
32 16.426857443  218.248.47.25   10.0.2.15       TCP        60 443 → 40564 [ACK] Seq=5407 Ack=1265 Win=65535 Len=0
33 16.509751139  218.248.47.25   10.0.2.15       TCP     10274 443 → 40564 [ACK] Seq=5407 Ack=1265 Win=65535 Len=10220 [TCP segment of a reassembled PDU]
34 16.509810229  10.0.2.15       218.248.47.25   TCP        54 40564 → 443 [ACK] Seq=1265 Ack=15627 Win=58400 Len=0

› Frame 29: 129 bytes on wire (1032 bits), 129 bytes captured (1032 bits) on interface eth0, id 0
› Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_2e:c5:1c (08:00:27:2e:c5:1c)
› Internet Protocol Version 4, Src: 218.248.47.25, Dst: 10.0.2.15
› Transmission Control Protocol, Src Port: 443, Dst Port: 40564, Seq: 5332, Ack: 668, Len: 75
▾ Transport Layer Security
  ▾ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.2 (0x0303)
      Length: 1
      Change Cipher Spec Message
```

Change Cipher Spec from Client

Step 7. Encrypted Handshake Message (Finished)

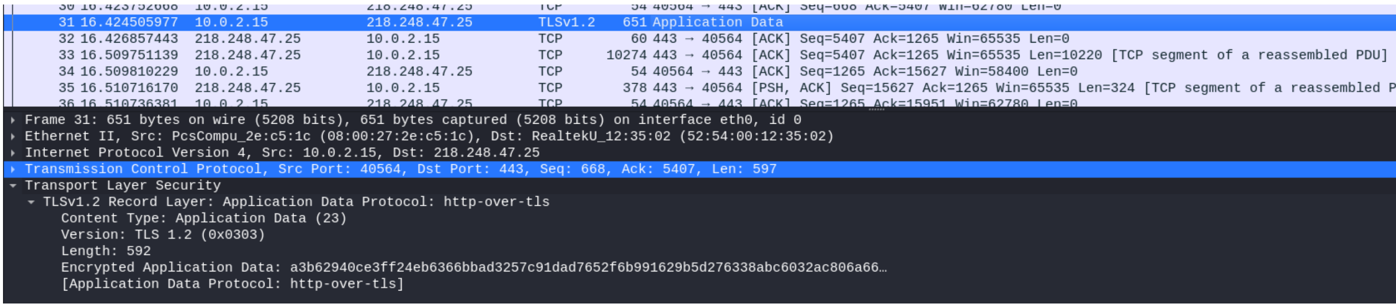
Like the Client Finished message but contains a different Label("server finished"). Once the client successfully decrypts and validates the message, the server is successfully authenticated.



Server Finished message encrypted

Application Data:

Once the TLS handshake is complete, Application Data is exchanged between the Client and the Server. The data is secure and encrypted with the agreed rules between them.



Encrypted application data

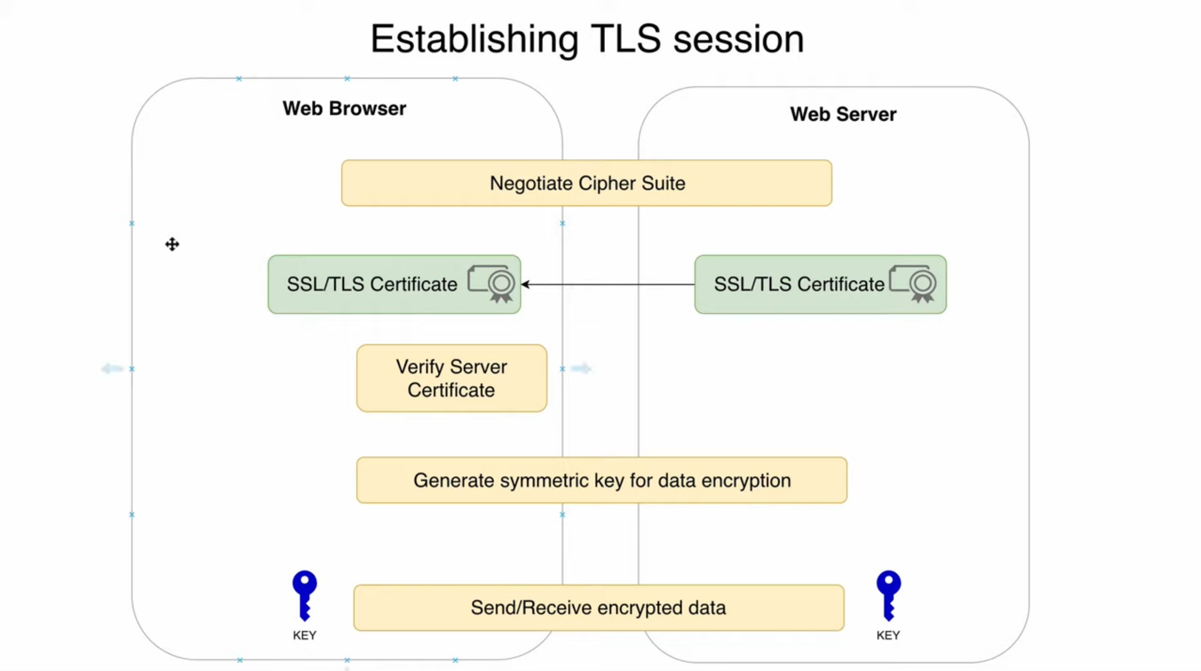
CIPHER SUITES

WHAT IS A CIPHER SUITE?

Cipher suites are sets of instructions that enable secure network connections through Transport Layer Security (TLS), often still referred to as Secure Sockets Layer (SSL). Behind the scenes, these cipher suites provide a set of algorithms and protocols required to secure communications between clients and servers.

Cryptographic security protocols must agree on the algorithms used by a secure connection. CipherSpecs and CipherSuites define specific combinations of algorithms. To initiate an HTTPS connection, the two parties – the web server and the client – perform an TLS handshake. The handshake process is a fairly complicated one, during which the two parties agree on a mutual cipher suite during *ClientHello* and *ServerHello*. The cipher suite is then used to negotiate a secure HTTPS connection.

WHY ARE CIPHER SUITES IMPORTANT?



TLS Session with cipher suites

Cipher suites are important for ensuring the security, compatibility, and performance of HTTPS connections. Just like recipes describe the required ingredients to make the perfect recipe, cipher suites dictate which algorithms to use to make a secure and reliable connection.

Both ends of a TLS, or SSL, the connection must agree on the same CipherSpec to be able to communicate and encrypt their information with the same algorithms.

These ciphers are required at various points of the connection to perform authentication, key generation and exchange. To determine what specific algorithms to use, the client and the web server start by mutually deciding on the cipher suite to be used.

WHAT ARE THE BASIC ELEMENTS OF A CIPHER SUITE?

The algorithms that make up a typical cipher suite are the following:

- **Key exchange algorithm** - dictates how symmetric keys will be exchanged. Some key exchange algorithms: RSA, DH, ECDH, ECDHE.
- **Authentication algorithm** - dictates how server authentication and (if needed) client authentication will be carried out. Some authentication algorithms: RSA, DSA, ECDSA.
- **Bulk encryption algorithm** - dictates which symmetric key algorithm will be used to encrypt the actual data. Some bulk encryption algorithms: AES, 3DES, CAMELLIA, (GCM, CBC Modes)

- **Message Authentication Code (MAC) algorithm** - dictates the method the connection will use to carry out data integrity checks. Some MAC algorithms: SHA, MD5.

  Key exchange algorithms protect information required to create shared keys. These algorithms are asymmetric (public key algorithms) and perform well for relatively small amounts of data.

  Bulk encryption algorithms encrypt messages exchanged between clients and servers. These algorithms are symmetric and perform well for large amounts of data.

  Message authentication algorithms generate message hashes and signatures that ensure the integrity of a message.

  Several algorithms are available for key exchange and authentication, but the RSA algorithm is currently the most widely used. There is more variety in the encryption algorithms and MAC algorithms that are used.

  Each cipher suite has a hexadecimal representation of its contents, which can be easily seen in the packet details in Wireshark.

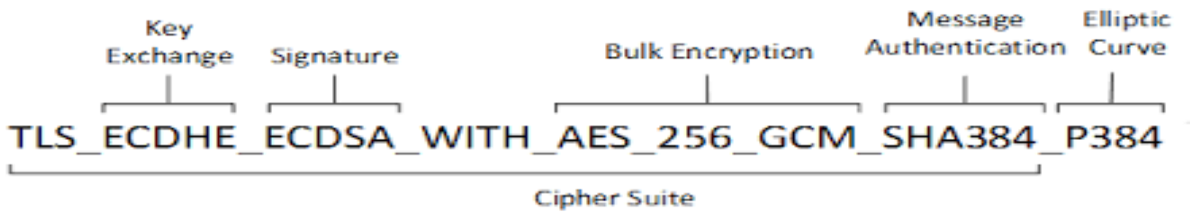
```
00 20 cc a8 cc a9 c0 2f c0 30 c0 2b c0 2c c0 13 c0 09 c0 14 c0 0a 00 9c 00 9d
00 2f 00 35 c0 12 00 0a
```

Cipher Suites

The client provides an ordered list of which cryptographic methods it will support for key exchange, encryption with that exchanged key, and message authentication. The list is in the order preferred by the client, with highest preference first.

- `00 20` - 0x20 (32) bytes of cipher suite data
- `cc a8` - assigned value for `TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256`
- `cc a9` - assigned value for `TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256`
- `c0 2f` - assigned value for `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- `c0 30` - assigned value for `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
- `c0 2b` - assigned value for `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
- `c0 2c` - assigned value for `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`
- `c0 13` - assigned value for `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA`
- `c0 09` - assigned value for `TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA`
- `c0 14` - assigned value for `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`
- `c0 0a` - assigned value for `TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA`
- `00 9c` - assigned value for `TLS_RSA_WITH_AES_128_GCM_SHA256`
- `00 9d` - assigned value for `TLS_RSA_WITH_AES_256_GCM_SHA384`
- `00 2f` - assigned value for `TLS_RSA_WITH_AES_128_CBC_SHA`
- `00 35` - assigned value for `TLS_RSA_WITH_AES_256_CBC_SHA`
- `c0 12` - assigned value for `TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA`
- `00 0a` - assigned value for `TLS_RSA_WITH_3DES_EDE_CBC_SHA`

Cipher Suites and their hexadecimal equivalents



Example of a typical cipher suite

WHEN ARE CIPHER SUITES USED?

Before a client application and a server can exchange data over an SSL/TLS connection, these two parties must agree on a common set of algorithms to secure the connection. If the two parties fail to reach an agreement, then a connection won't be established.

The negotiation process takes place during what is commonly known as the SSL handshake. In the SSL handshake, the client begins by informing the server what cipher suites it supports. The cipher suites are usually arranged in order of security. The most secure cipher suite naturally becomes the first choice.

The server then compares those cipher suites with the cipher suites that are enabled on its side. As soon as it finds a match, it informs the client, and the chosen cipher suite's algorithms are called into play.

CHOOSING CIPHER SUITES

TLS 1.3 cipher suites are not interoperable with older TLS versions because their structure is different. This means that site administrators will need to configure their web servers in such a way to allow compatibility with both versions of supported cipher suites, TLS 1.2 and TLS 1.3. Opting for support of only TLS 1.3 is not a wise solution, because a lot of companies rely still on TLS 1.2.

IDENTIFYING WEAK CIPHERS

With the introduction of TLS 1.3, many things changed to improve the security of the protocol. To start with, old, insecure ciphers have been deprecated, including:

- RC4
- DSA
- MD5
- SHA-1
- Weak Elliptic Curves
- RSA Key Exchange
- Static Diffie-Hellman (DH, ECDH)
- Block ciphers (CBC)
- Non-AEAD ciphers

- **ANALYSIS**

In our analysis of TLS handshaking, the following cipher suites were sent by the client to the server in ClientHello.

```
    Cipher Suites Length: 34
  ▾ Cipher Suites (17 suites)
      Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
      Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
      Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
      Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
      Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
      Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
      Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
      Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
```

<u>List of cipher suits sent by client to server in ClientHello</u>

```
Session ID: d508000089200b23a27bbff84d6ee5ed372c68bd23de09e9e4df0bea00116312
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
Compression Method: null (0)
```

<u>Cipher Suite selected by the server in ServerHello</u>

The server selected Cipher Suite which is more secure and can be used by both, the client and the server to establish a symmetric encrypted connection.

In this case, the server selected TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA

- TLS: Simply indicates the protocol, i.e. Transport Layer Security Protocol.
- ECDHE: Key Exchange Algorithm, i.e.. Elliptic Curve Diffie Hellman Ephemeral Algorithm
- RSA: Authentication Algorithm, i.e.. Rivest–Shamir–Adleman Algorithm.
- AES_256_CBC: Bulk Cipher Encryption Algorithm, i.e.. Advanced Encryption Standard using a 256-bit key and mode CBC (cipher block chaining) Mode.
- SHA: MAC Algorithms, SHA-384 (Secure Hash algorithm) is a patented cryptographic hash function that outputs a value that is 256 bits long.

This is mutually agreed upon by both the client and the server and once the Encrypted Handshake Message (Finished) is passed by both client and server, all future messages will be encrypted and secure by the same algorithm on both sides and decrypted on the other side.

CONTRIBUTION SUMMARY

HARSHIKA SOFAT & ANUSHKA DAS – The project was done together with each part taken care by both the team members. We worked on Kali Linux on one of our laptop and executed all the necessary steps. One documented the TLS handshake part whereas the other documented the Cipher suits.

REFERENCES

1. https://www.internetsociety.org/deploy360/tls/basics
2. https://www.cloudflare.com/en-in/learning/ssl/what-happens-in-a-tls-handshake/
3. https://www.ibm.com/docs/en/ibm-mq/7.5?topic=ssl-ciphersuites-cipherspecs
4. https://tls.ulfheim.net/
5. https://docs.microsoft.com/en-us/windows/win32/secauthn/cipher-suites-in-schannel
6. https://www.keyfactor.com/blog/cipher-suites-explained/
7. https://www.venafi.com/blog/what-are-cipher-suites
8. https://www.jscape.com/blog/cipher-suites
9. https://clienttest.ssllabs.com:8443/ssltest/viewMyClient.html

THANK YOU