교과목 : 정보보호

10. Transport Layer Security

2023학년도 2학기
Suk-Hwan Lee

- References
  - ✓ William Stallings, Cryptography and Network Security, 7th edition
  - ✓ W. Shbair, Service-Level Monitoring of HTTPS Traffic, Univ. of Luxembourg, 2017
  - ✓ York Univ. N. Viajic, Network Security, Lecture Note, 2019
  - ✓ Cyprus Univ., IIT Madras "Transport Layer Security" Lecture Note
  - ✓ 순천향대 암호와 네트워크 보안 강의자료 참조
  - ✓ Wikipedia : https://en.wikipedia.org/wiki/Transport_Layer_Security

- The World Wide Web is fundamentally a client/server application running over the Internet and TCP/IP intranets
- The following characteristics of Web usage suggest the need for tailored security tools:
  - ✓ Web servers are relatively easy to configure and manage
  - ✓ Web content is increasingly easy to develop. The underlying software is extraordinarily complex
    - ✓ May hide many potential security flaws
  - ✓ A Web server can be exploited as a launching pad into the corporation’ s or agency’ s entire computer complex
  - ✓ Casual and untrained (in security matters) users are common clients for Web-based services

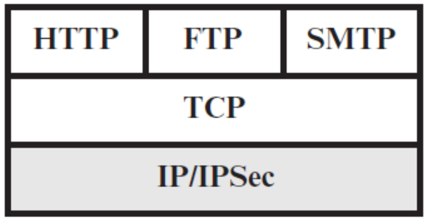
Web Security Threats

Comparison of Threats on the Web

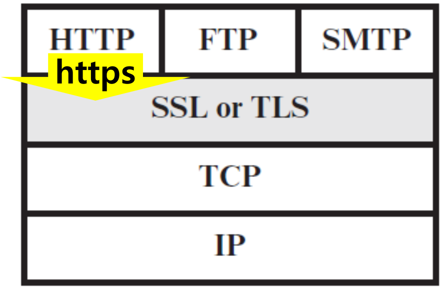
	Threats	Consequences	Countermeasures
Integrity	• Modification of user data • Trojan horse browser • Modification of memory • Modification of message traffic in transit	• Loss of information • Compromise of machine • Vulnerability to all other threats	Cryptographic checksums
Confidentiality	• Eavesdropping on the net • Theft of info from server • Theft of data from client • Info about network configuration • Info about which client talks to server	• Loss of information • Loss of privacy	Encryption, Web proxies
Denial of Service	• Killing of user threads • Flooding machine with bogus requests • Filling up disk or memory • Isolating machine by DNS attacks	• Disruptive • Annoying • Prevent user from getting work done	Difficult to prevent
Authentication	• Impersonation of legitimate users • Data forgery	• Misrepresentation of user • Belief that false information is valid	Cryptographic techniques

Web Security Approaches

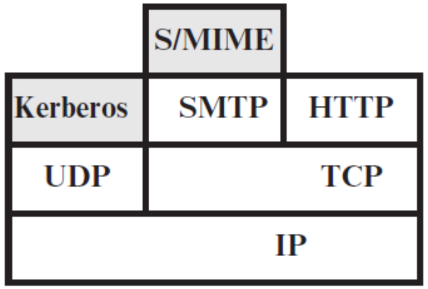
- Relative location of security facilities in TCP/IP Protocol stack



Network level



Transport level



Application level

Network Layer Security
Deploy IPSec at the network layer
- ✓ Transparent to end users and applications and provides a general-purpose solution.
- ✓ IPSec : 통신 세션의 각 IP패킷을 암호화하고 인증

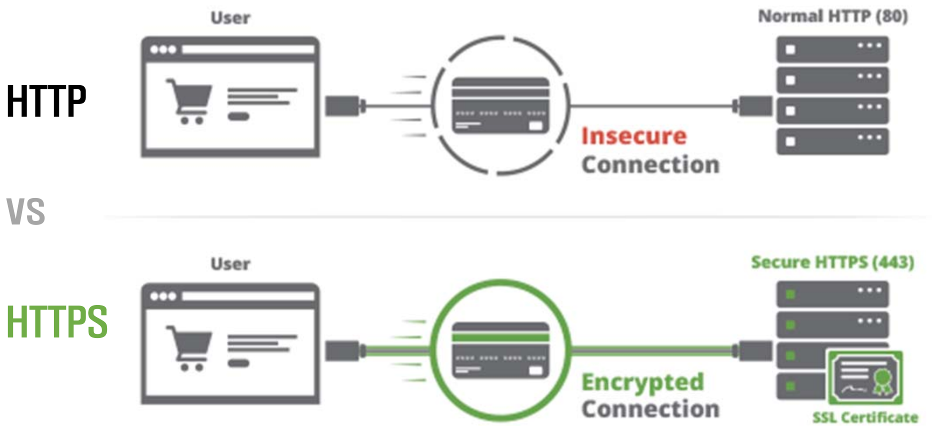
Just Above TCP Security – SSL/TLS
Keep TCP/IP 'as is', add protection on top of TCP
- ✓ Cryptographic protocols designed to provide communications security over a computer network
- ✓ Provide privacy and data integrity between two or more communicating computer applications

Application-specific security
- ✓ Tailored to the specific needs of a given application
- ✓ S/MIME (Secure for Multipurpose Internet Mail Extensions) : MIME 객체에 임호화와 전자서명 추가
- ✓ Kerberos : 네트워크 인증 암호화 프로토콜 (티켓 기반으로 비보안 네트워크에 특정 노드와 노드가 보안된 형식으로 통신하도록 제공)

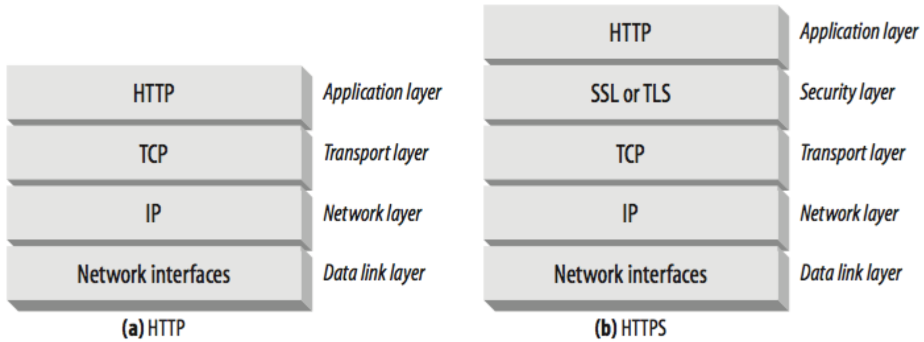
TLS Key Application

HTTPS = HTTP over TLS

- Fixes the problem of standard HTTP which transmits data in plaintext (more later)



[발췌] https://www.suntech.org.ng/2018/08/09/the-importance-of-getting-an-ssl-certificate-on-your-website/



[발췌] https://heidyhe.github.io/https/

Usage of Default protocol https for websites, 7 Jan 2020, W3Techs.com

(W3Techs 통계) 2020년 1월 기준 전세계 웹사이트의 57.5%가 HTTPS 프로토콜을 사용

2020년 1월 7일 현재 https 사용 웹사이트 점유율 (출처: W3Techs)

TLS :
Protocol to achieve secure communication

- TLS provide secure communication channel with 3 properties:
  - ✓ Confidentiality
  - ✓ Integrity
  - ✓ Authentication
- Two important components
  - ✓ TLS Handshake
  - ✓ Secure Data Communication

SSL vs TLS History

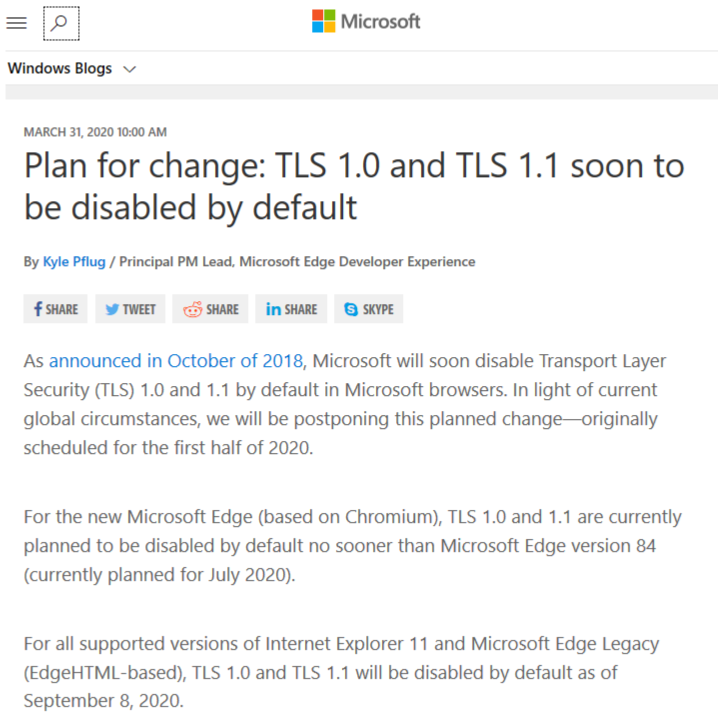
- 1995 : Netscape released SSL 2.0
- 1996 : New version SSL 3.0
- 1999 : TLS introduced as the new version of SSL (TLS 1.0)
- 2011 : SSL 2.0 deprecated by IETF(국제인터넷표준기구)
- 2015 : SSL 3.0 deprecated by IETF
- 2006 : TLS 1.1 → deprecated in Jan. 2019
- 2008 : TLS 1.2
- 2018 : TLS1.3

❖ Difference : Handshake protocols changes from SSL to TLS and Encryption

[참고]

Microsoft TLS1.0, TLS1.1 비활성화

- Chromium 기반 Microsoft Edge(2020년1월15일 릴리즈)의 경우 기본적으로 비활성화
- 또한 Internet Explorer 11과 Microsoft Edge Legacy (EdgeHTML-based) 버전은 2020년 9월 8일 부터 기본적으로 비활성화 될 예정
- MS는 브라우저 및 웹 사이트를 서비스를 하는 서버에서 TLS 1.0 및 TLS 1.1에 대해 비활성화와 TLS 1.2이상을 사용할 것을 권고하고 있습니다.


Microsoft

Windows Blogs

MARCH 31, 2020 10:00 AM

Plan for change: TLS 1.0 and TLS 1.1 soon to be disabled by default

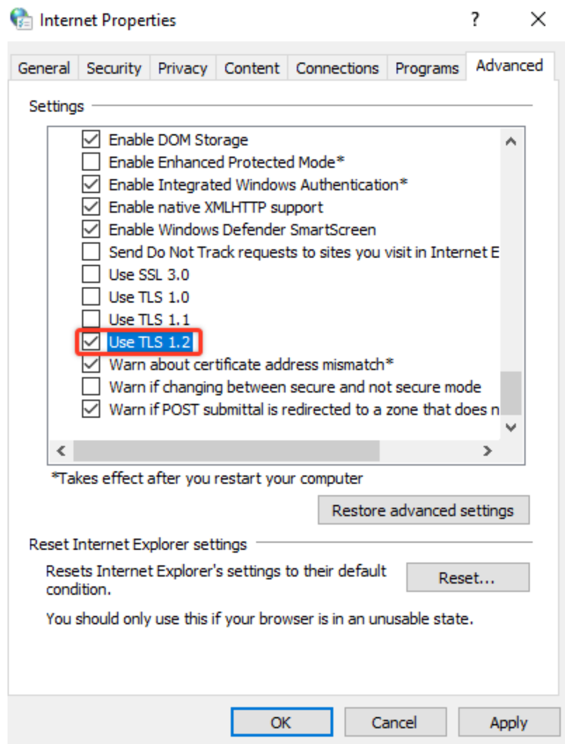
By Kyle Pflug / Principal PM Lead, Microsoft Edge Developer Experience

SHARE TWEET SHARE SHARE SKYPE

As announced in October of 2018, Microsoft will soon disable Transport Layer Security (TLS) 1.0 and 1.1 by default in Microsoft browsers. In light of current global circumstances, we will be postponing this planned change—originally scheduled for the first half of 2020.

For the new Microsoft Edge (based on Chromium), TLS 1.0 and 1.1 are currently planned to be disabled by default no sooner than Microsoft Edge version 84 (currently planned for July 2020).

For all supported versions of Internet Explorer 11 and Microsoft Edge Legacy (EdgeHTML-based), TLS 1.0 and TLS 1.1 will be disabled by default as of September 8, 2020.



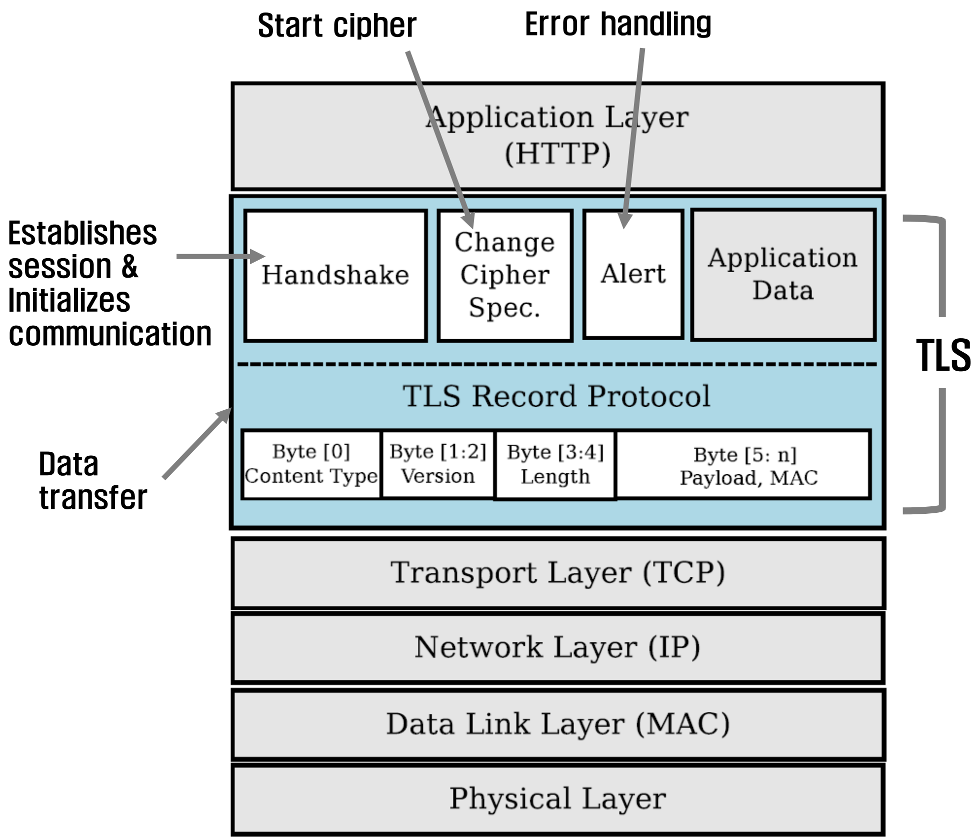
브라우저의 옵션설정에서 TLS 1.0 및 TLS 1.1 프로토콜을 활성화시킬 수 있음

TLS architecture

- 2 layers of 4 protocols
- Top-layer
  - ✓ Handshake Protocol : Provides security parameters for Record Protocol – establish connection
  - ✓ ChangeCipherSpec Protocol : Signals readiness of cryptographic secrets – establish connection
  - ✓ Alert Protocol : Report abnormal conditions
- Lower-layer
  - ✓ Record Protocol : Carries message from other 3 protocols as well as application data

➢ 2 important TLS concepts

TLS connection, TLS session



TLS layers and sub-protocols

TLS Session

- An association between a client and a server. Created by the Handshake Protocol.
- After a session is established, two parties have common information (session state parameters) exchanged, including:

Parameter	Description
Session ID	A server-chosen 8-bit number defining a session
Peer Certificate	A certificate of type X.509.v3. This parameter may be empty (null)
Compression method	The compression method
Cipher Suite	The agreed-upon cipher suite; encryption (null, AES, etc), hash (MD5 or SHA-1) used for MAC calculation
Master Secret	The 48-byte secret shared between the client and server
Is resumable	A flag indicating whether the session can be used to initiate the new connections

For two entities to exchange data,
the establishment of a session is necessary, but not sufficient!
They also need to create a connection between themselves!

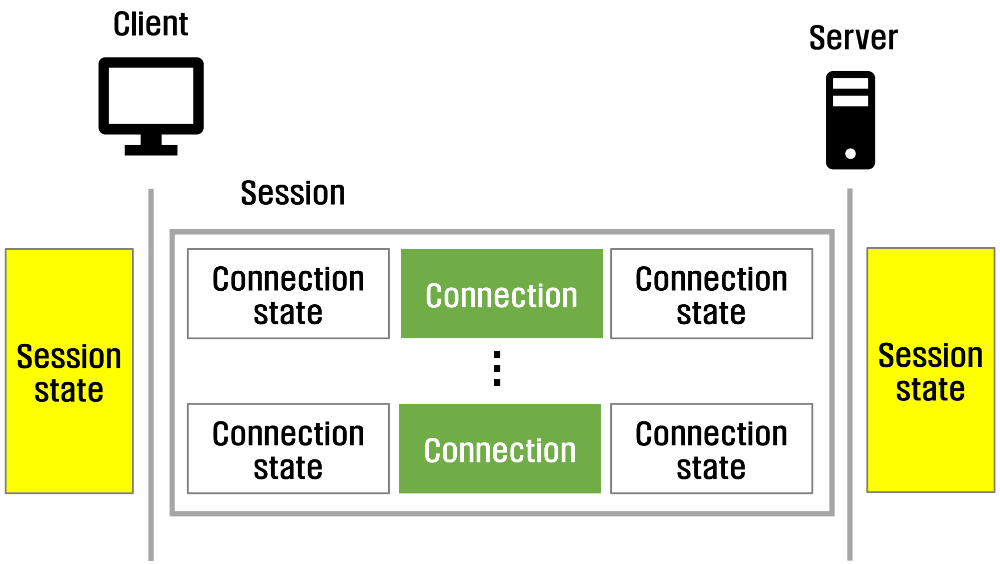
TLS Session vs Connections

- A session can consist of many connections
  - ✓ a connection between two parties can be terminated and reestablished within the same session
  - ✓ a session can be suspended and resumed again
  - ✓ to resume an old session and create a new connection, two parties can skip part of negotiation and go through a shorter one – there is no need to create a master secret when a session is resumed

Separation of session from connection prevents the high cost of creating master secret!

In a session, one party has the role of a client and the other the role of a server.

In a connection, both parties have equal roles – they are peers



TLS Connections

- To establish a connection, and actually be able to exchange data, two entities have to exchange two random numbers and create, using master secret, the read and write keys and parameters – (so-called connection state parameters)

The client and the server have 6 different cryptography secrets–– 3 read and 3 write secrets. The read secrets for the client are the same as write secrets for the server and vice versa.

Parameter	Description
Server and client random exchanged	Byte sequences chosen by the server and client for each connection.
Server write MAC secret	The outbound serve MAC key for message integrity. The server uses it to sign; the client uses it to verify
Client write MAC secret derived	The outbound client MAC key for message integrity. The client uses it to sign; the server uses it to verify
Server write key	The outbound server encryption key
Client write key	The outbound client encryption key
Initialization vectors	The block ciphers in CBC model use initialization vectors (Ivs). One initialization vector is defined for each cipher key during the negotiation, which is used for the first block exchange. The final cipher text from a block is used as the IV for the next block
Sequence numbers	Each party has a sequence numbers. The sequence number starts from 0 and increments. It must not exceed $2^{64}-1$

Symmetric key for data encrypted by server and decrypted by client

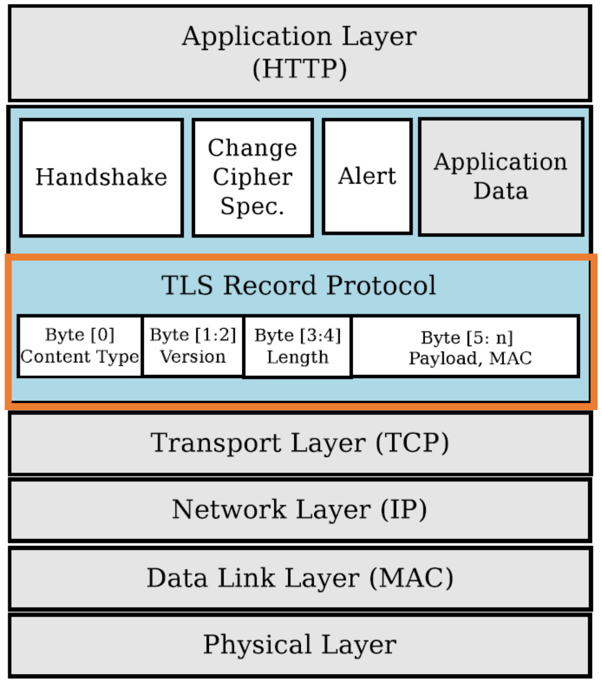
Symmetric key for data encrypted by client and decrypted by server

TLS Record Protocol

TLS Record protocol responsible for

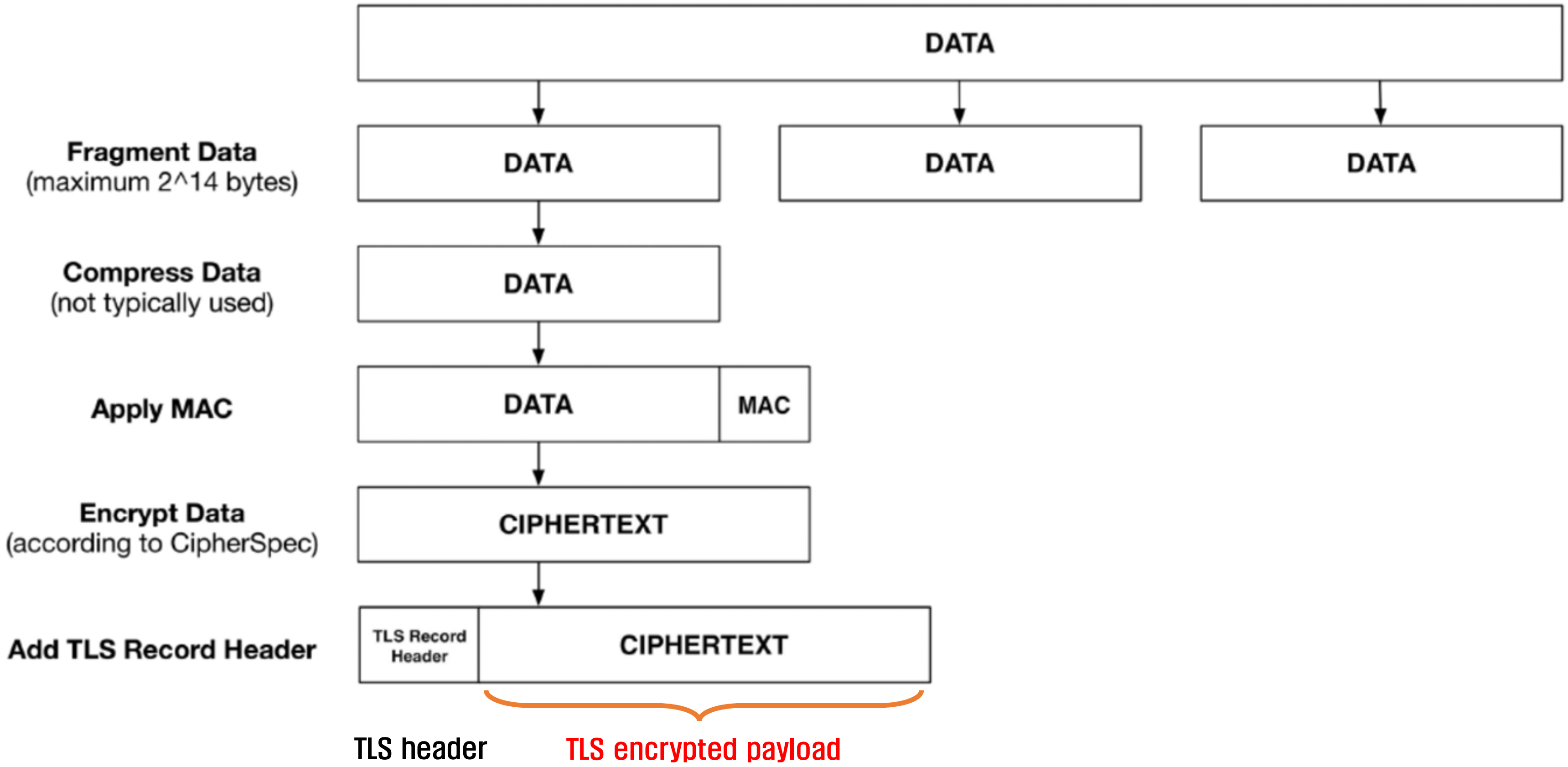
- ensuring confidentiality of application data
- verifying its integrity & integrity of its origin; specific roles include:

1) fragmenting higher-layer protocol data into blocks of 214bytes or less
2) optionally compressing data
3) adding Message Authentication Code (MAC)
4) encrypting data
5) adding TLS record header



TLS Record Protocol

- Example : Sending data with TLS Record Protocol – Processing

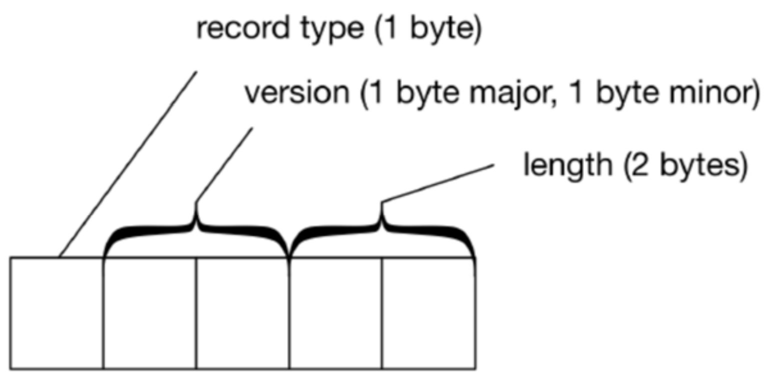


1) TLS header

record = packet of Record Protocol – consist of header (TLS header) & payload (TLS payload)

- records are not used only for transfer of application data: message in ChangeCipherSpec, Handshake and Alert Protocol are also transferred using records

- there are 3 fields of TLS header



TLS Record Header

+	Byte +0	Byte +1	Byte +2	Byte +3
Byte 0	Content type			
Bytes 1..4	Legacy version		Length	
	(Major)	*(Minor)*	*(bits 15..8)*	*(bits 7..0)*
Bytes 5..(*m*−1)	Protocol message(s)			
Bytes *m*..(*p*−1)	MAC (optional)			
Bytes *p*..(*q*−1)	Padding (block ciphers only)			

1) TLS header

record = packet of Record Protocol – consist of header (TLS header) & payload (TLS payload)

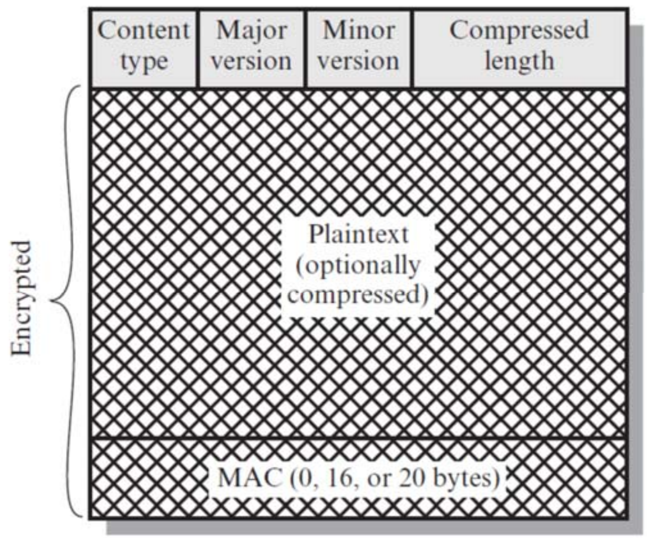


Figure 17.4 TLS Record Format

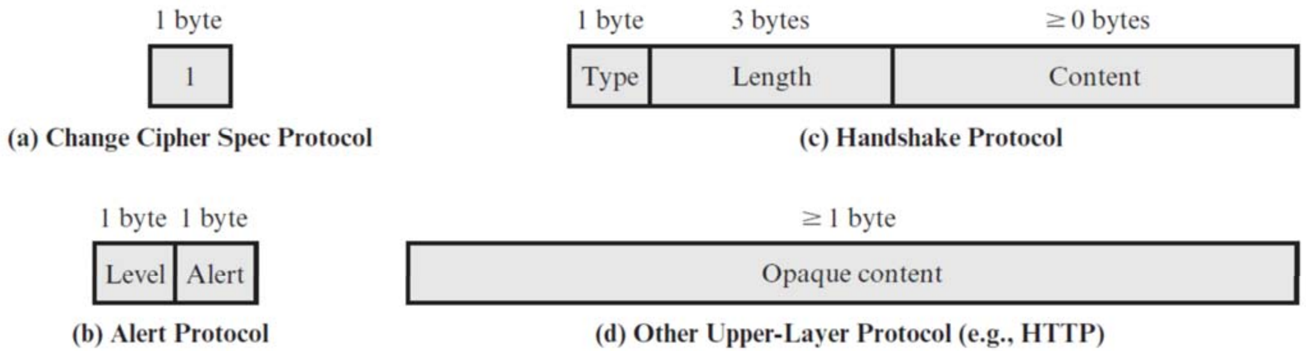


Figure 17.5 TLS Record Protocol Payload

TLS Record Protocol

1) TLS header

3 fields of TLS header

Byte +0	Byte +1	Byte +2	Byte +3
Content type			
Version		Length	
(Major)	*(Minor)*	*(bits 15..8)*	*(bits 7..0)*

1) content type (record type) : 1byte long field;

Indicates what **type of protocol data** is carried by the current record

Hex	Dec	Type
0x14	20	ChangeCipherSpec
0x15	21	Alert
0x16	22	Handshake
0x17	23	Application
0x18	24	Heartbeat

2) Version : 2byte long field; identifies the major (1byte) and minor version (1byte) of TLS for the given message

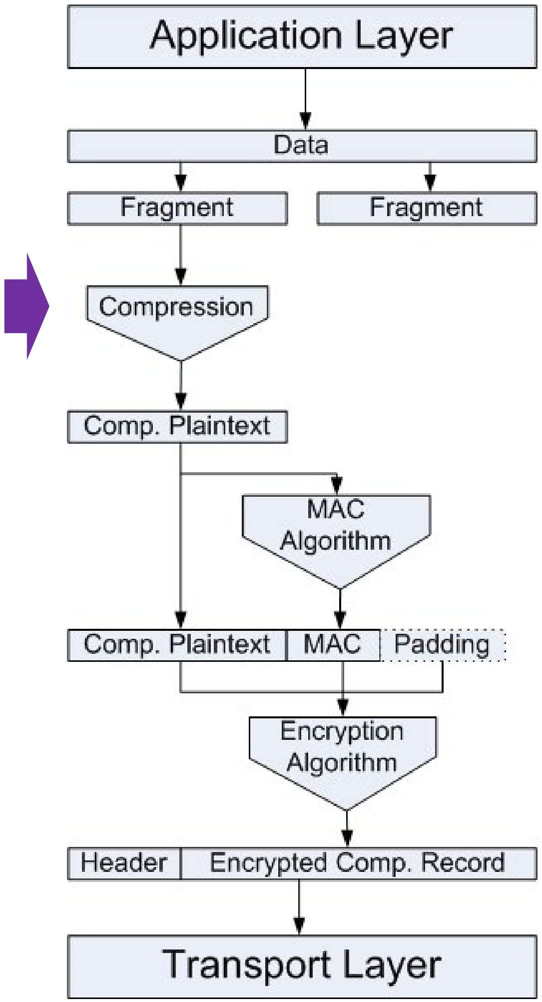
Major version	Minor version	Version type
3	0	SSL 3.0
3	1	TLS 1.0
3	2	TLS 1.1
3	3	TLS 1.2
3	4	TLS 1.3

3) Length : 2-byte long field; identifies the **length of the payload field** (MAC and padding combined) in bytes,

should not exceed 2^{14}+ 2048 bytes

TLS Record Protocol

2) Compression



- Optionally applied, must be lossless
- most common algorithm : DEFLATE (FC 3749) (ZIP, gzip 등의 프로그램에서 사용되는 무손실 압축 데이터 포맷이자 알고리즘)

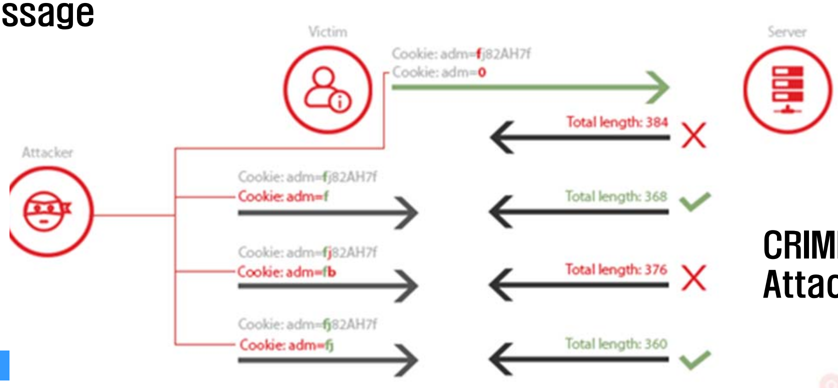
❖ Note

- Dangers of using TLS Compression
  - ✓ To date, numerous attacks exploiting TLS compression have been identified: CRIME, TIME, BREACH …

Recommendation : Disable TLS compression

Wireshark capture : Client Hello Message

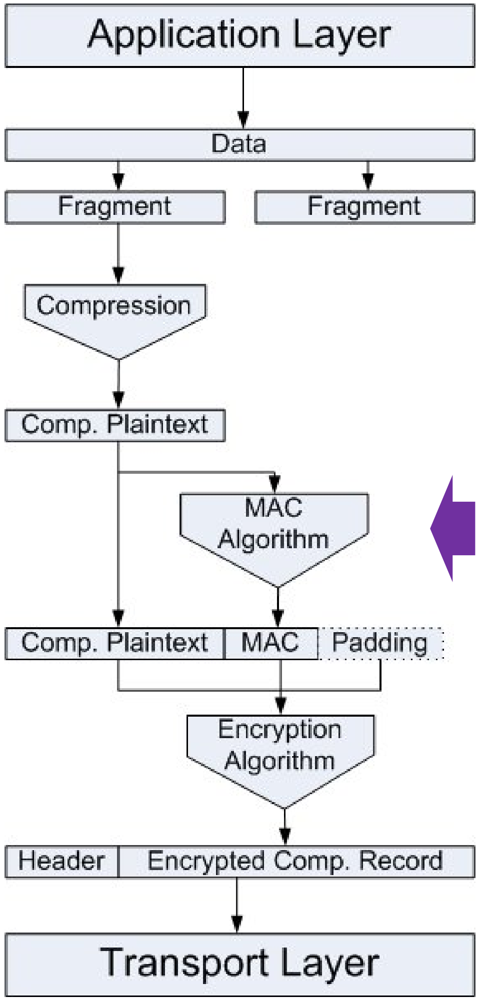
```
⊟ Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 103
    Version: TLS 1.0 (0x0301)
  ⊞ Random
    Session ID Length: 0
    Cipher Suites Length: 40
  ⊞ Cipher Suites (20 suites)
    Compression Methods Length: 2
  ⊟ Compression Methods (2 methods)
      Compression Method: DEFLATE (1)
      Compression Method: null (0)
```



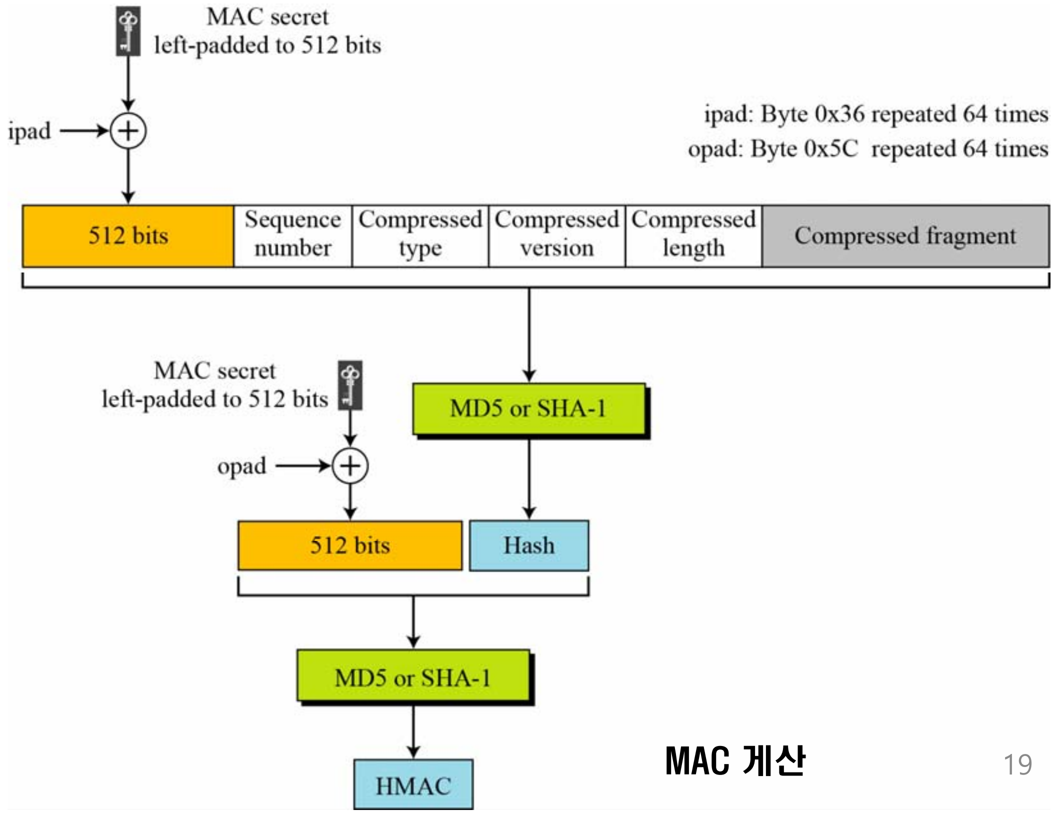
CRIME Attack

출처 : https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/

3) TLS Hashing



- TLS makes use of HMAC algorithm to compute Message Authentication Code (MAC) over data
  - ➢ hash algorithms used: HMAC_MD5, HMAC_SHA1, HMAC_SHA256, HMAC_SHA384, HMAC_SHA512



MAC 계산

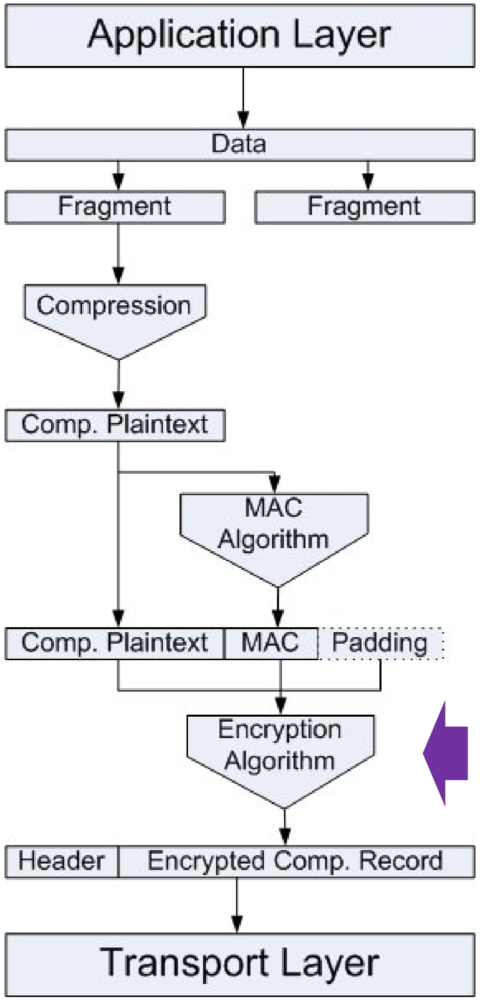
3) TLS Hashing

Notes : Different Types of MAC algorithms in TLS

Data integrity

Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	RFC status
HMAC-MD5	Yes	Yes	Yes	Yes	Yes	No	Defined for TLS 1.2 in RFCs
HMAC-SHA1	No	Yes	Yes	Yes	Yes	No	
HMAC-SHA256/384	No	No	No	No	Yes	No	
AEAD	No	No	No	No	Yes	Yes	
GOST 28147-89 IMIT[53]	No	No	Yes	Yes	Yes		Proposed in RFC drafts
GOST R 34.11-94[53]	No	No	Yes	Yes	Yes		

4) TLS Encryption



- Compressed message plus the MAC are encrypted using symmetric encryption
  - ✓ Encryption may not increase content length by more than 1024 bytes, so that the total length may not exceed 2^{14}+2048 bytes
  - ✓ Keys for this symmetric encryption are generated uniquely for each connection and are based on a secret negotiated by TLS Handshake Protocol
  - ✓ However, Record Protocol can be used without encryption
  - ✓ For stream encryption, compressed message + MAC are encrypted
  - ✓ For block encryption, padding may be added after MAC and prior to encryption, in order to result in blocks of data that are multiple of cipher's block length, up to a maximum of 255 bytes

TLS Record Protocol

Notes : TLS Ciphers

Cipher security against publicly known feasible attacks

Cipher			Protocol version						
Type	Algorithm	Nominal strength (bits)	SSL 2.0	SSL 3.0 [n 1][n 2][n 3][n 4]	TLS 1.0 [n 1][n 3]	TLS 1.1 [n 1]	TLS 1.2 [n 1]	TLS 1.3	Status
Block cipher with mode of operation	AES GCM[54][n 5]	256, 128	N/A	N/A	N/A	N/A	Secure	Secure	Defined for TLS 1.2 in RFCs
	AES CCM[55][n 5]		N/A	N/A	N/A	N/A	Secure	Secure	
	AES CBC[n 6]		N/A	Insecure	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A	
	Camellia GCM[56][n 5]	256, 128	N/A	N/A	N/A	N/A	Secure	N/A	
	Camellia CBC[57][n 6]		N/A	Insecure	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A	
	ARIA GCM[58][n 5]	256, 128	N/A	N/A	N/A	N/A	Secure	N/A	
	ARIA CBC[58][n 6]		N/A	N/A	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A	
	SEED CBC[59][n 6]	128	N/A	Insecure	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A	
	3DES EDE CBC[n 6][n 7]	112[n 8]	Insecure	Insecure	Insecure	Insecure	Insecure	N/A	
	GOST 28147-89 CNT[53][n 7]	256	N/A	N/A	Insecure	Insecure	Insecure	N/A	Defined in RFC 4357
	IDEA CBC[n 6][n 7][n 9]	128	Insecure	Insecure	Insecure	Insecure	N/A	N/A	Removed from TLS 1.2
	DES CBC[n 6][n 7][n 9]	56	Insecure	Insecure	Insecure	Insecure	N/A	N/A	
		40[n 10]	Insecure	Insecure	Insecure	N/A	N/A	N/A	Forbidden in TLS 1.1 and later
	RC2 CBC[n 6][n 7]	40[n 10]	Insecure	Insecure	Insecure	N/A	N/A	N/A	
Stream cipher	ChaCha20-Poly1305[64][n 5]	256	N/A	N/A	N/A	N/A	Secure	Secure	Defined for TLS 1.2 in RFCs
	RC4[n 11]	128	Insecure	Insecure	Insecure	Insecure	Insecure	N/A	Prohibited in all versions of TLS by RFC 7465
		40[n 10]	Insecure	Insecure	Insecure	N/A	N/A	N/A	
None	Null[n 12]	–	Insecure	Insecure	Insecure	Insecure	Insecure	N/A	Defined for TLS 1.2 in RFCs

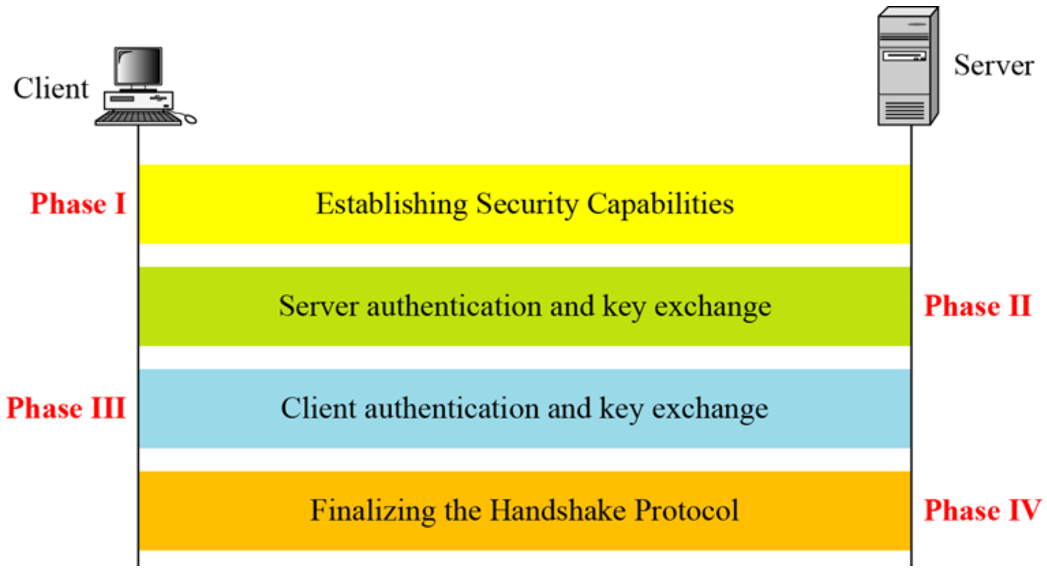
TLS Handshake Protocol

- Most complex part of TLS – achieves multiple objectives:

> - assists client & server in agreeing on TLS version used
> - authenticates client & server to each other
> - negotiates MAC algorithms, encryption algorithms and keys used to protect data sent in TLS record

Handshake Protocol is used before any application data is transmitted!

It is responsible for negotiation of security parameters used by TLS Record Layer!



4 main phases of SSL/TLS Handsake

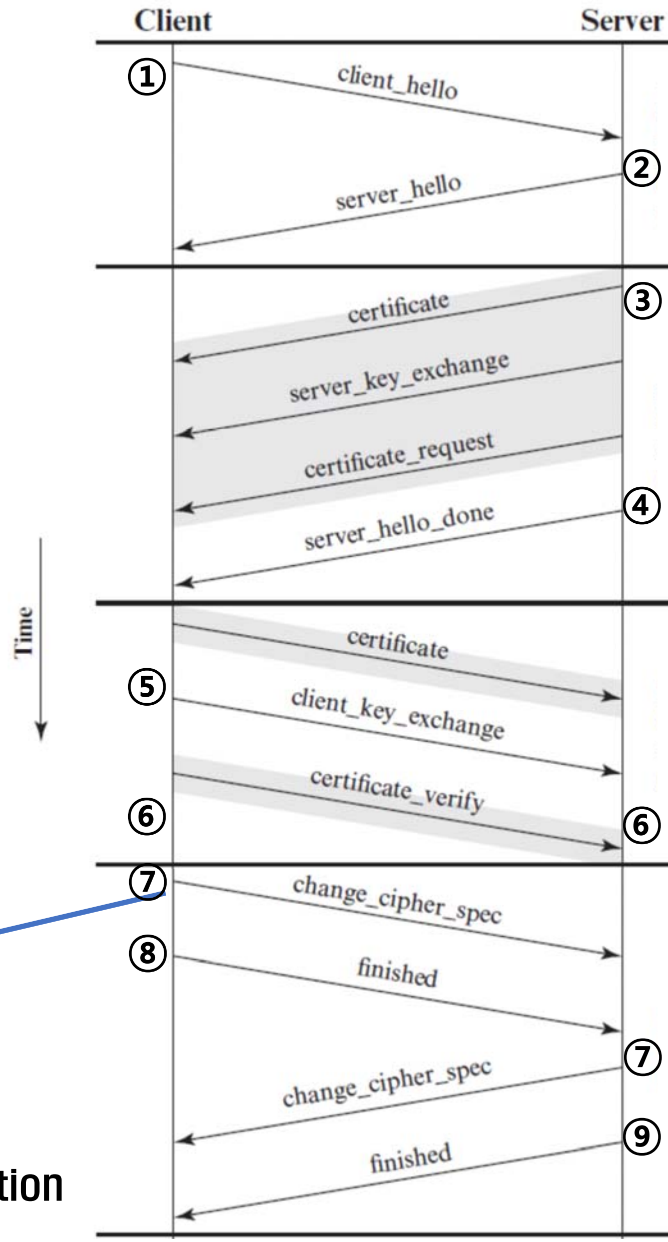
TLS Handshake Protocol

TLS Handshake Protocol

In TLS the values exchanged between C and S are not sufficient to generate the symmetric key (s).

C and S exchange only parameters that are later used to generate the actual key(s).

An important message – signifies the beginning of encryption!



Handshake Protocol Action

Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.

Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.

Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.

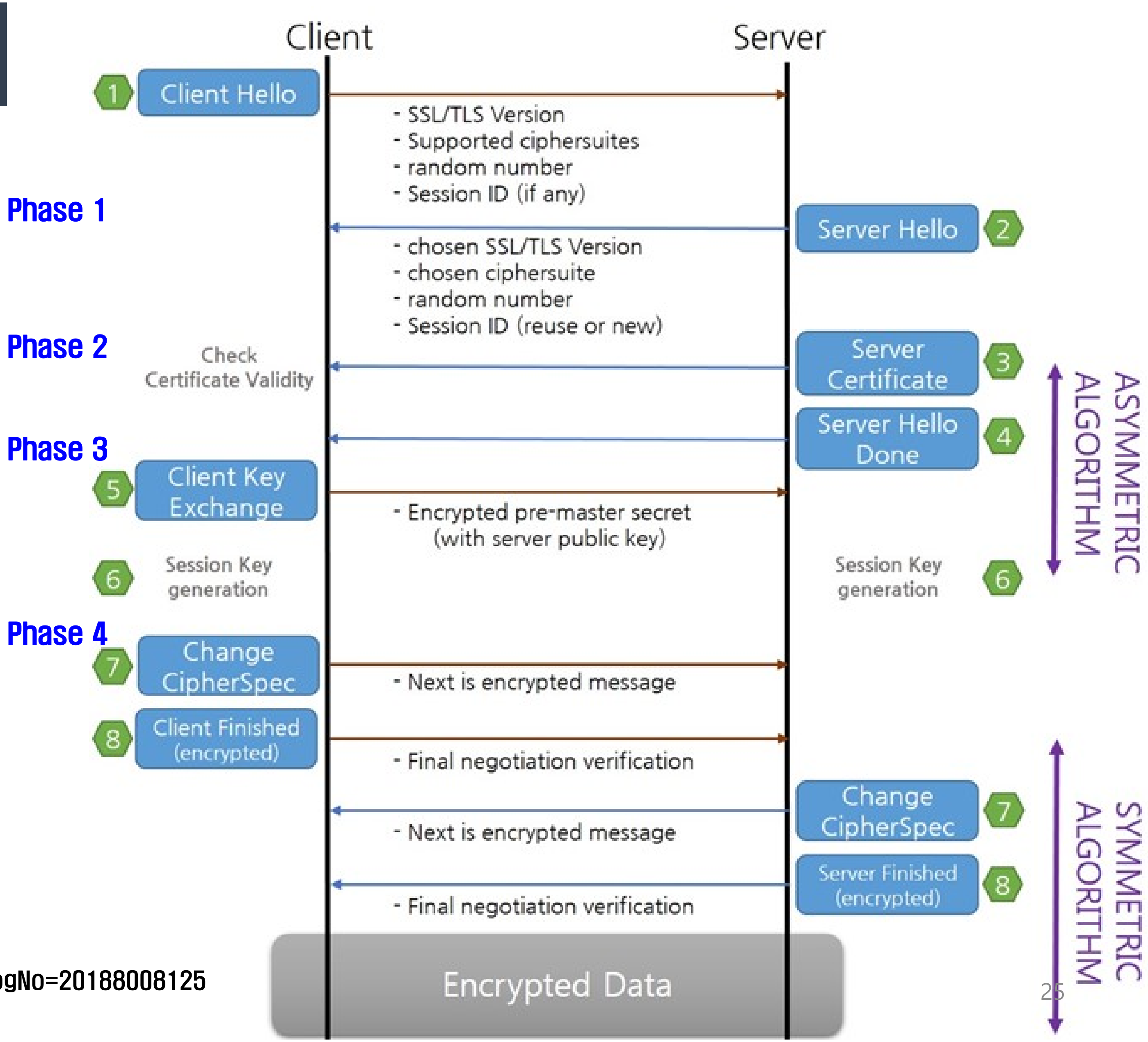
Note: Shaded transfers are optional or situation-dependent messages that are not always sent

Phase 4
Change cipher suite and finish handshake protocol.

TLS Handshake Protocol

TLS Handshake Protocol

Handshake Protocol Message Sequence



[출처]
http://blog.naver.com/PostView.nhn?blogId=seri0528&logNo=20188008125

TLS Handshake Protocol

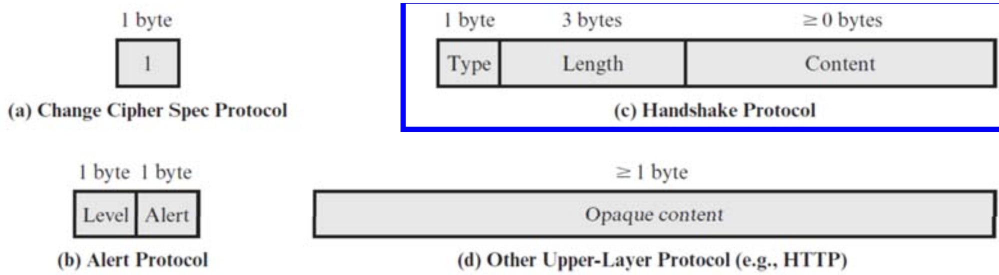


Figure 17.5 TLS Record Protocol Payload

TLS Handshake Packet Format

Record Protocol Header

Handshake Protocol message1

Handshake Protocol message2

+	Byte +0	Byte +1	Byte +2	Byte +3
Byte 0	22			
Bytes 1..4	Version		Length	
	(Major)	(Minor)	(bits 15..8)	(bits 7..0)
Bytes 5..8	Message type	Handshake message data length		
		(bits 23..16)	(bits 15..8)	(bits 7..0)
Bytes 9..(*n*−1)	Handshake message data			
Bytes *n*..(*n*+3)	Message type	Handshake message data length		
		(bits 23..16)	(bits 15..8)	(bits 7..0)
Bytes (*n*+4)..	Handshake message data			

One Record Protocol packet can carry multiple Handshake protocol messages

Hex	Dec	Type
0x14	20	ChangeCipherSpec
0x15	21	Alert
0x16	22	Handshake
0x17	23	Application
0x18	24	Heartbeat

- **Type (1 byte)**: Indicates one of 10 messages. *Table 17.2 lists the defined message types.*
- **Length (3 bytes)**: The length of the message in bytes.
- **Content (# 0 bytes)**: The parameters associated with this message; *these are listed in Table 17.2. [다음 페이지]*

Message types	
Code	**Description**
0	HelloRequest
1	ClientHello
2	ServerHello
4	NewSessionTicket
8	EncryptedExtensions (TLS 1.3 only)
11	Certificate
12	ServerKeyExchange
13	CertificateRequest
14	ServerHelloDone
15	CertificateVerify
16	ClientKeyExchange
20	Finished

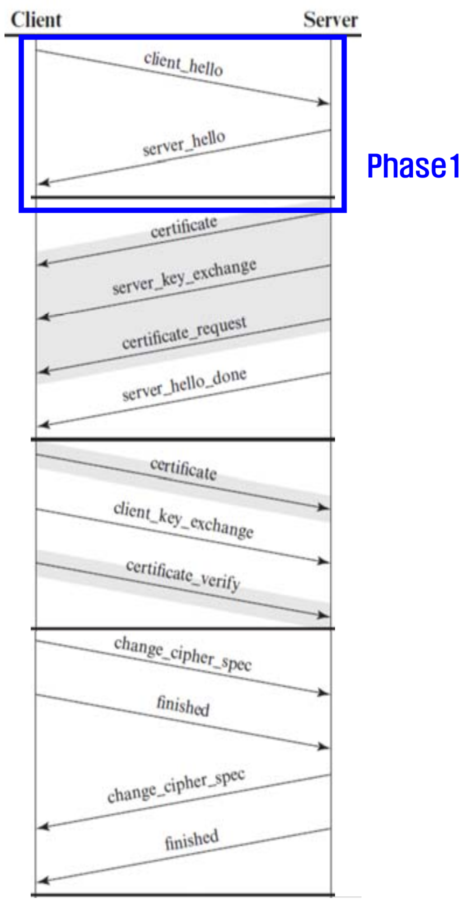
TLS Handshake Packet Format

Message types	
Code	**Description**
0	HelloRequest
1	ClientHello
2	ServerHello
4	NewSessionTicket
8	EncryptedExtensions (TLS 1.3 only)
11	Certificate
12	ServerKeyExchange
13	CertificateRequest
14	ServerHelloDone
15	CertificateVerify
16	ClientKeyExchange
20	Finished

Table 17.2 TLS Handshake Protocol Message Types

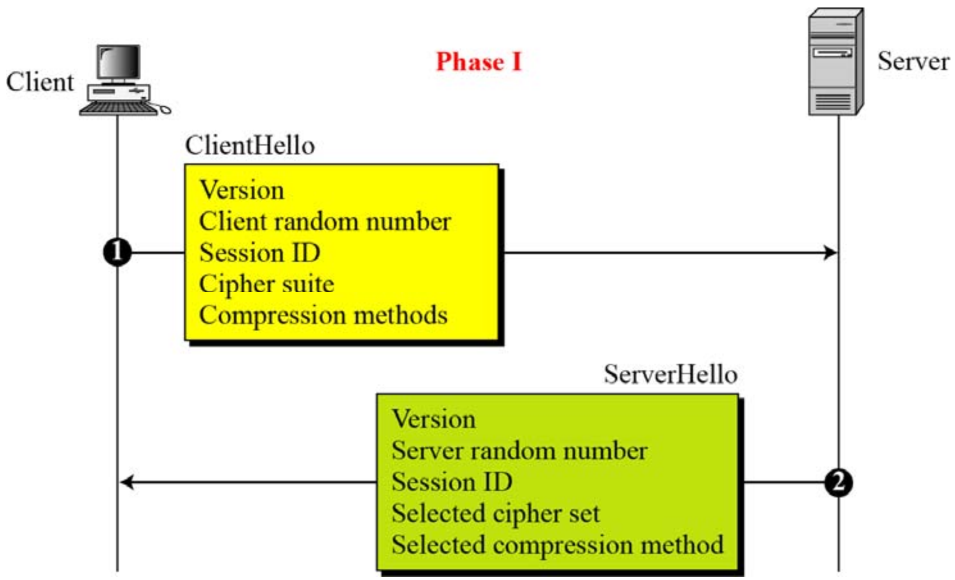
Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

Handshake Phase 1 – Establishing Security Capability



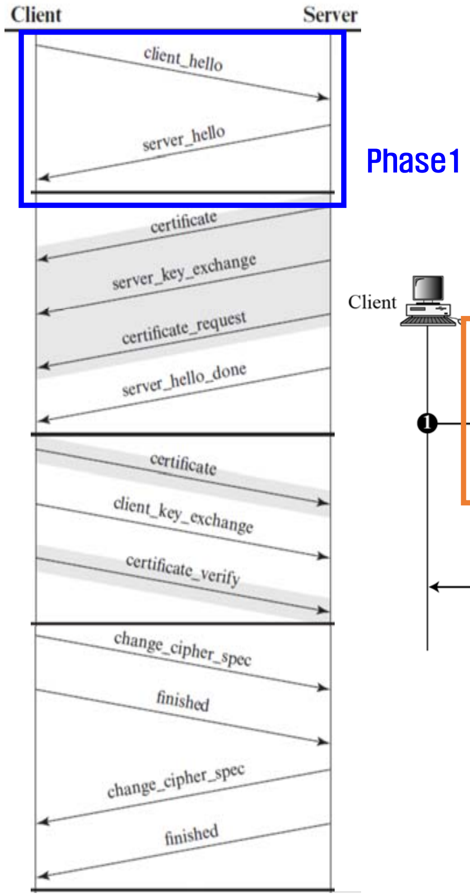
= client & server announce their security capabilities and choose those that are convenient for both

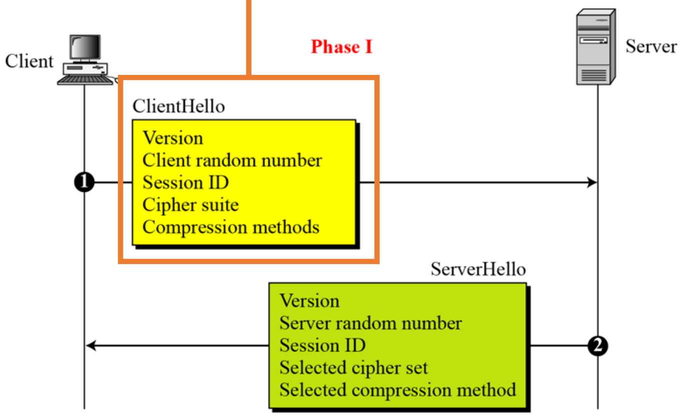
- ✓ in addition to agreeing on: TLS version, algorithms for key exchange, message authentication and encryption, compression method, & session IDs, 2 random numbers are also exchanged by client and server which are then used to create 'master secret'
- ✓ initiated by the client, which sends ClientHello message



- ✓ After sending ClientHello message, C waits for ServerHello message, which contains the same parameters as ClientHello message

Handshake Phase 1 – Establishing Security Capability

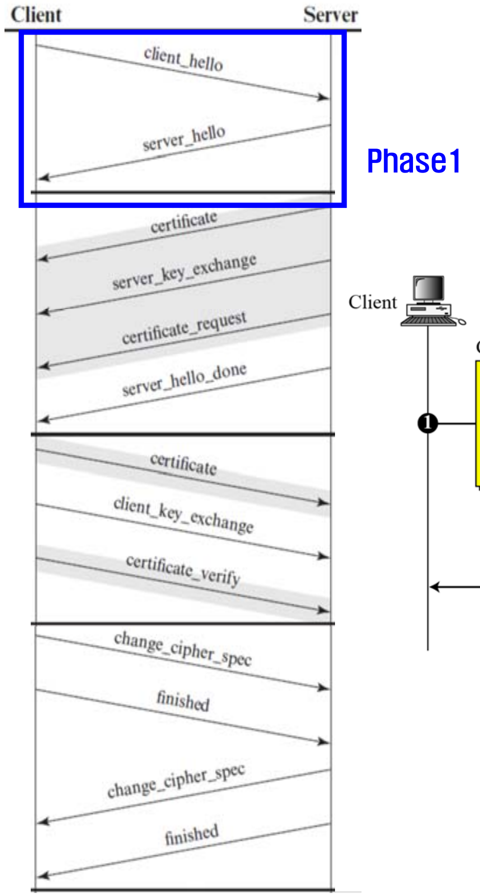


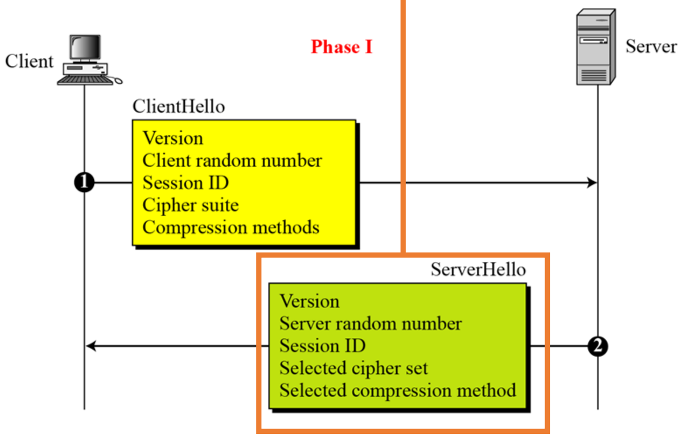


ClientHello message contains:

- ✓ **version**= highest TLS version # the client can support
- ✓ **random**= 32-bit timestamp & 28 bytes generated by a secure random number generator to serve as nonces to prevent replay attacks and are used during pre-master key exchange
- ✓ **session ID**= variable length session identifier
  - a non-zero value indicates that client wishes to update the parameters of an existing connection or to create a new connection on this session
  - a zero value indicates that client wishes to establish a new connection on a new session
- ✓ **CipherSuite**= list that contains cryptographic algorithms supported by the client
- ✓ **compression method**= list of compression methods that client can supports

Handshake Phase 1 – Establishing Security Capability

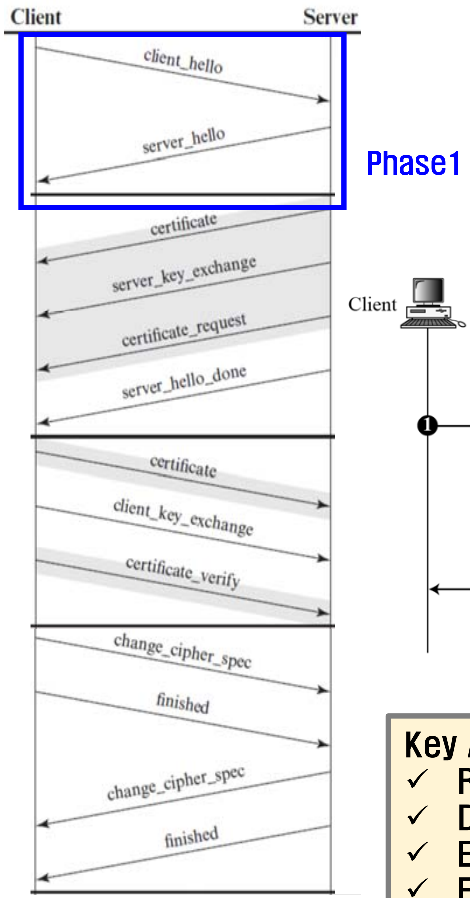


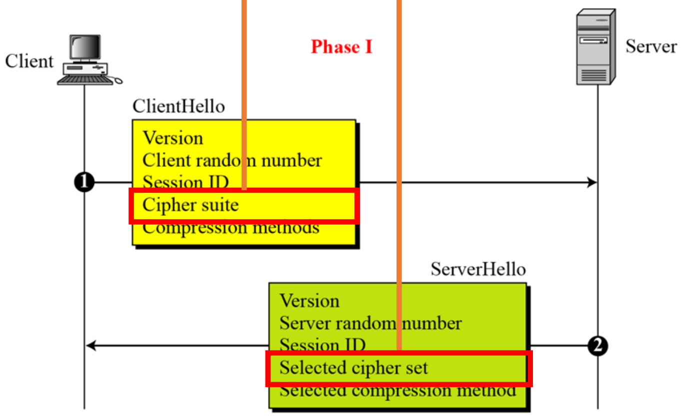


ServerHello message contains:

- ✓ version= the lower of two version numbers: the highest
- ✓ supported by client and highest supported by server
- ✓ random= same procedure as in case of Client Hello
- ✓ session ID=
  - if session ID sent by client is not zero, server will search for previously cached sessions and if a match is found, that session ID will be used;
  - otherwise a new session will be created, i.e., the server will return 0
- ✓ CipherSuite= single cipher suite selected by server from those proposed by client – if supported, server will agree on client' s preferred cipher suite
- ✓ compression method= compression method selected by server from those proposed by client – if supported, server will agree on client' s preferred compression method

Handshake Phase 1 – Establishing Security Capability





CipherSuite specifies an algorithm for each:

- ✓ public-key algorithm for creation of symmetric keys
- ✓ authentication algorithm
- ✓ symmetric algorithm for data encryption
- ✓ hash algorithm for data authentication

Key Agreement — Cipher + size + mode

TLS_ECDHE_RSA_WITH_AES_256 _GCM_SHA384

Protocol — Signature — HMAC + size

Key Agreement Available Algorithms:
- ✓ RSA
- ✓ Diffie-Hellman
- ✓ ECDH (Elliptic Curve DH)
- ✓ ECDHE (Elliptic Curve DH Ephemeral)

Authentication Algorithms:
- ✓ RSA
- ✓ DSA (Digital Signature Algorithm)
- ✓ ECDSA (Elliptic Curve DSA)

Symmetric Encryption Algorithms:
- ✓ AES, 3DES
- ✓ CAMELLIA, ARIA,
- ✓ SEED

MAC Algorithms:
- ✓ MD5
- ✓ SHA
- ✓ SHA256
- ✓ SHA384

Handshake Phase 1 – Establishing Security Capability

Example : TLS Cipher Suites

Many TLS Cipher Suites exist, but arbitrary combinations not possible – generally determined by the OS!

For Windows 10, version 1903, 1909, and 2004, the following cipher suites are enabled and in this priority order by default using the Microsoft Schannel Provider:

Cipher suite string	Allowed by SCH_USE_STRONG_CRYPTO	TLS/SSL Protocol versions
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	Yes	TLS 1.2
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	Yes	TLS 1.2
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Yes	TLS 1.2
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	Yes	TLS 1.2
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	No	TLS 1.2
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	Yes	TLS 1.2
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	Yes	TLS 1.2
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	Yes	TLS 1.2
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	Yes	TLS 1.2
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	Yes	TLS 1.2
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	Yes	TLS 1.2, TLS 1.1, TLS 1.0

출처 : https://docs.microsoft.com/en-us/windows/win32/secauthn/tls-cipher-suites-in-windows-10-v1903

Handshake Phase 1 – Establishing Security Capability

Example : TLS Cipher Suites from a Packet Capture

```
Secure Sockets Layer
  TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 186
   Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 182
      Version: TLS 1.2 (0x0303)
      Random
      Session ID Length: 0
      Cipher Suites Length: 22
      Cipher Suites (11 suites)
        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_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_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
        Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
        Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
        Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
        Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
      Compression Methods Length: 1
```

SSL/TLS versions supported by client

Cipher Suites supported by client

```
Secure Sockets Layer
  TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 70
   Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 66
      Version: TLS 1.2 (0x0303)
      Random
      Session ID Length: 0
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
      Compression Method: null (0)
      Extensions Length: 26
      Extension: server_name
      Extension: renegotiation_info
      Extension: ec_point_formats
      Extension: Application Layer Protocol Negotiation
```

SSL/TLS version and cipher suite picked by the server

출처 : https://www.linuxbabe.com/security/ssltls-handshake-process-explained-with-wireshark-screenshot

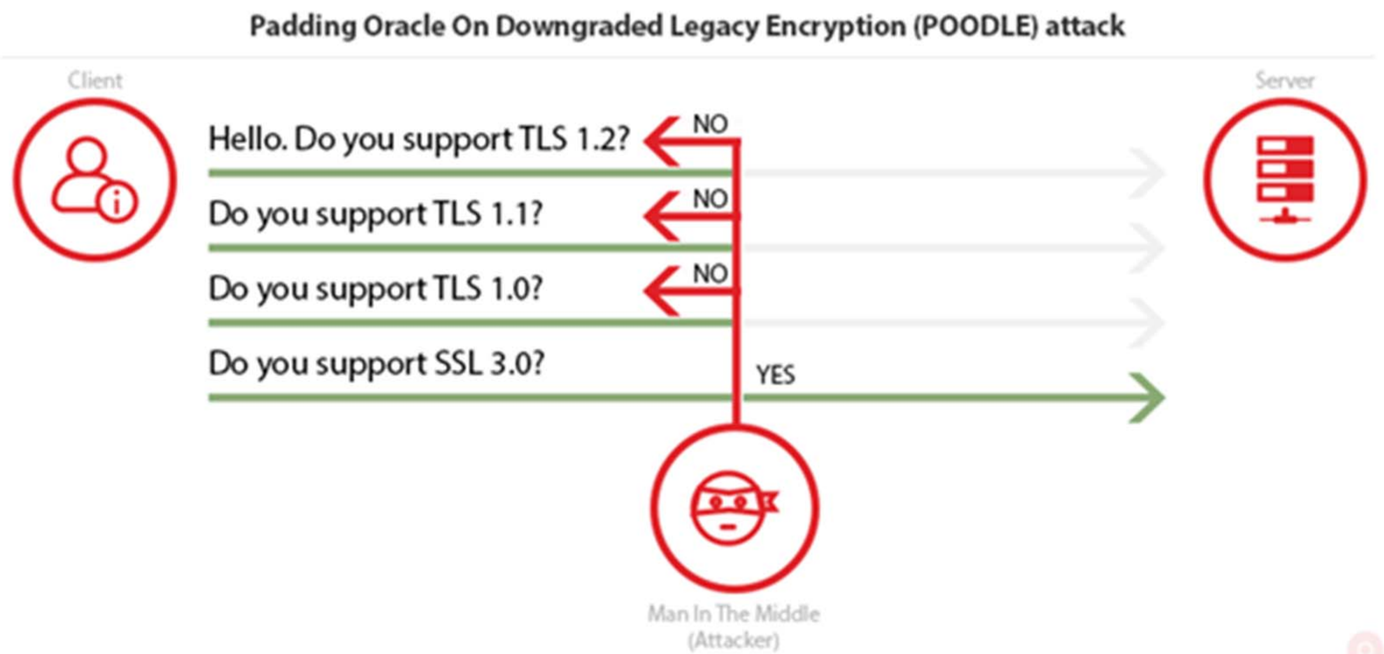
Handshake Phase 1 – Establishing Security Capability

Example : Downgrade Attack on TLS – a variant of MitM attack

Padding Oracle On Downgraded Legacy Encryption (POODLE)

attacker aims to downgrade entire TLS (not very sophisticated)

TLS 연결 설정과정에서 하위버전인 SSL3.0으로 연결 수립을 유도한 뒤, 패딩 오라클 공격을 통해 암호화된 통신내용을 복호화하는 공격기법

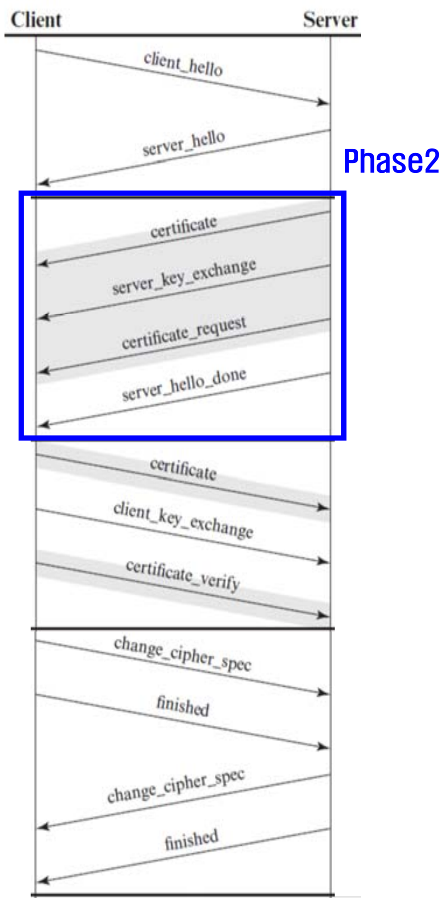


Prevention

- Completely disable SSL 3.0 on the server
- Upgrade the browser to the latest version

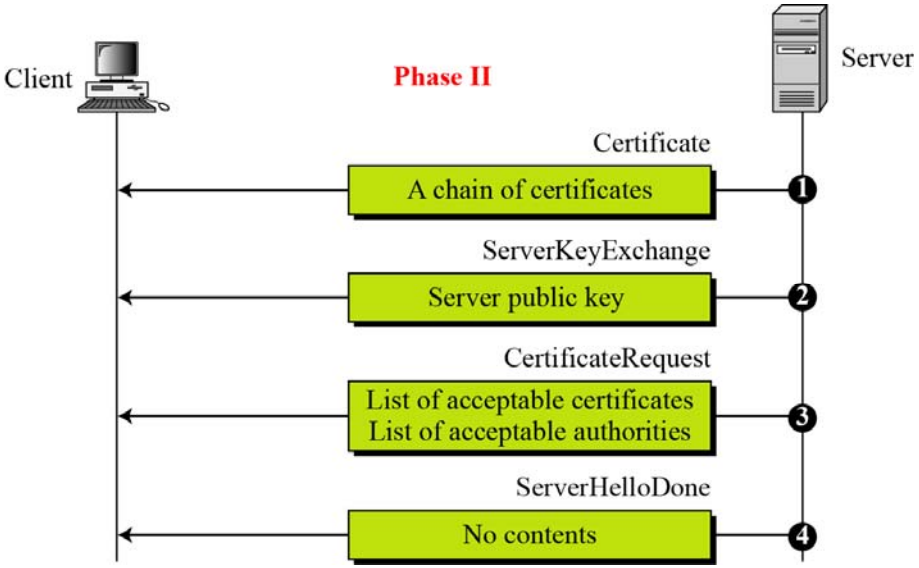
출처 : https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/

Handshake Phase 2 – Server Authentication & Key Exchange



= server sends its X.509 certificates (if needed), its public key, and may also request certificates from client

① Certificates= if required, server sends a list of certificates of type X.509 containing its own certificate & certificates of all intermediate CA's, in case they are not on client's list of trusted CAs (not needed if anonymous Diffie-Hellman key exchange performed)



Public-key encryption is computationally expensive. Thus, TLS uses public-key cryptography to establish the symmetric key which is later used for encryption of actual data.

Entire symmetric key generation consists of 3 steps:

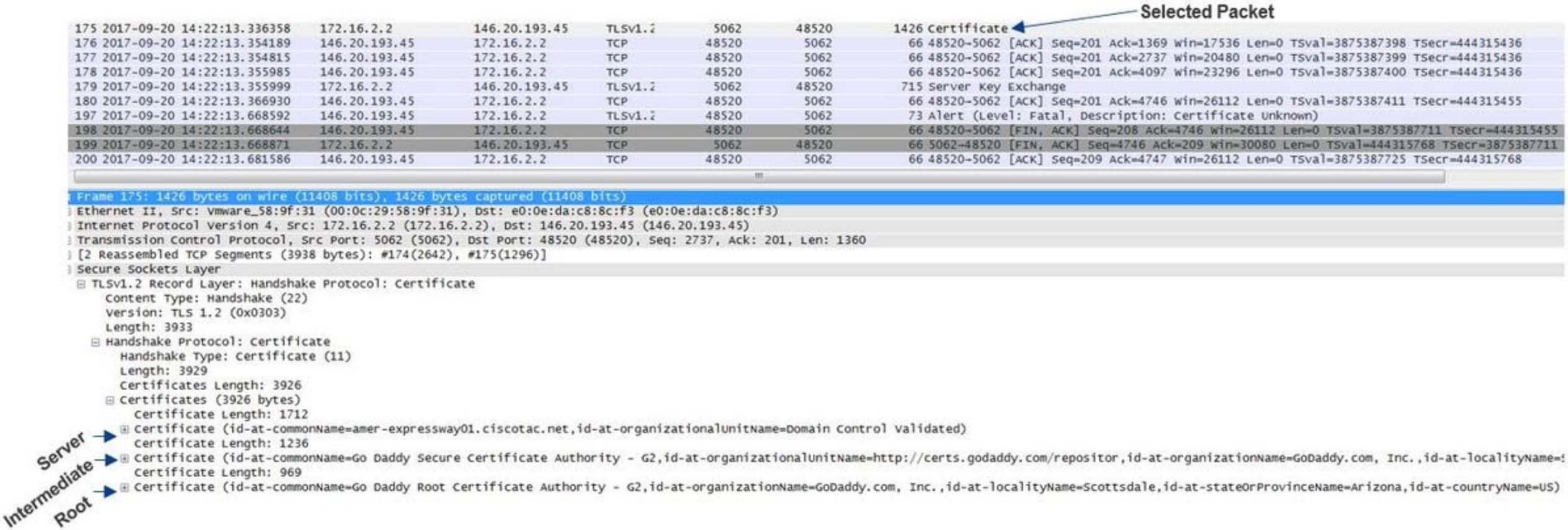
1) Generate pre-master secret

2) Generate master secret

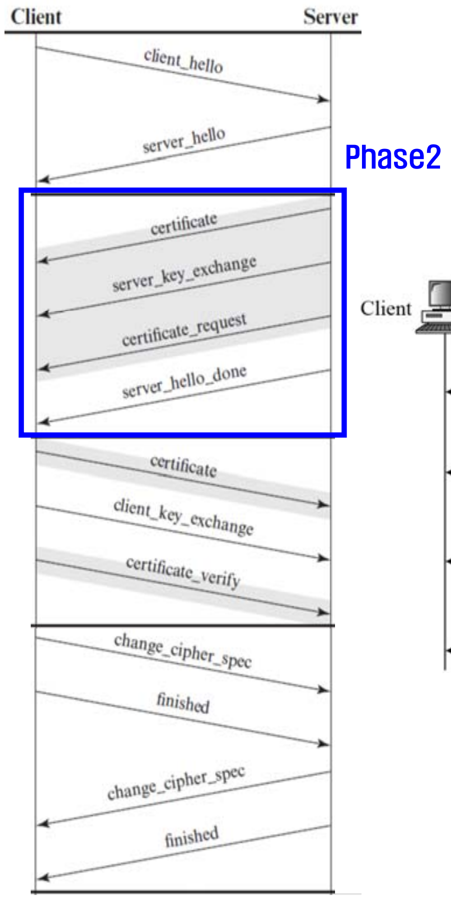
3) Generate session keys.

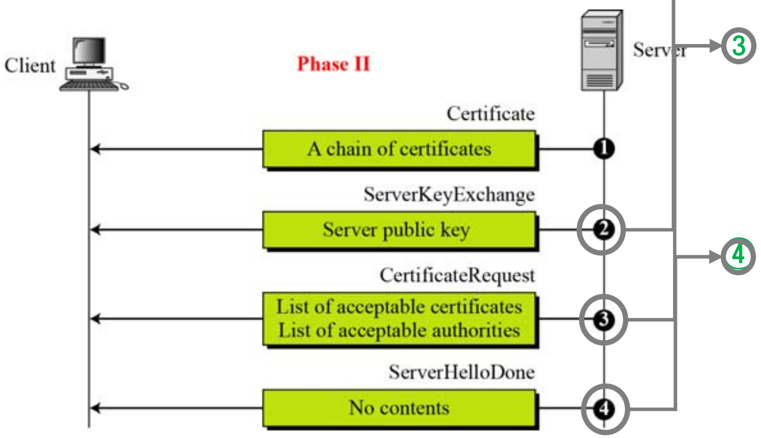
Handshake Phase 2 – Server Authentication & Key Exchange

Example : TLS Certificates in Packet Capture



Handshake Phase 2 – Server Authentication & Key Exchange





- ② **ServerKyeExchange**= after Certificates message, server sends a **ServerKeyExchange** message that includes its contribution to pre-master secret (not required if key – exchange method is RSA or fixed Diffie–Hellman)
- ③ **CertificateRequest**= sent if server needs to authenticate client itself (not common, and not required in case of anonymous Diffie–Hellman key exchange)
- ④ **ServerHelloDone**= signals the client that Phase II is over

Note : Pre-master Secret vs Master Secret vs Read/Write Keys

Pre-master secret:

value exchanged/generated by client and server – different approaches in different methods.

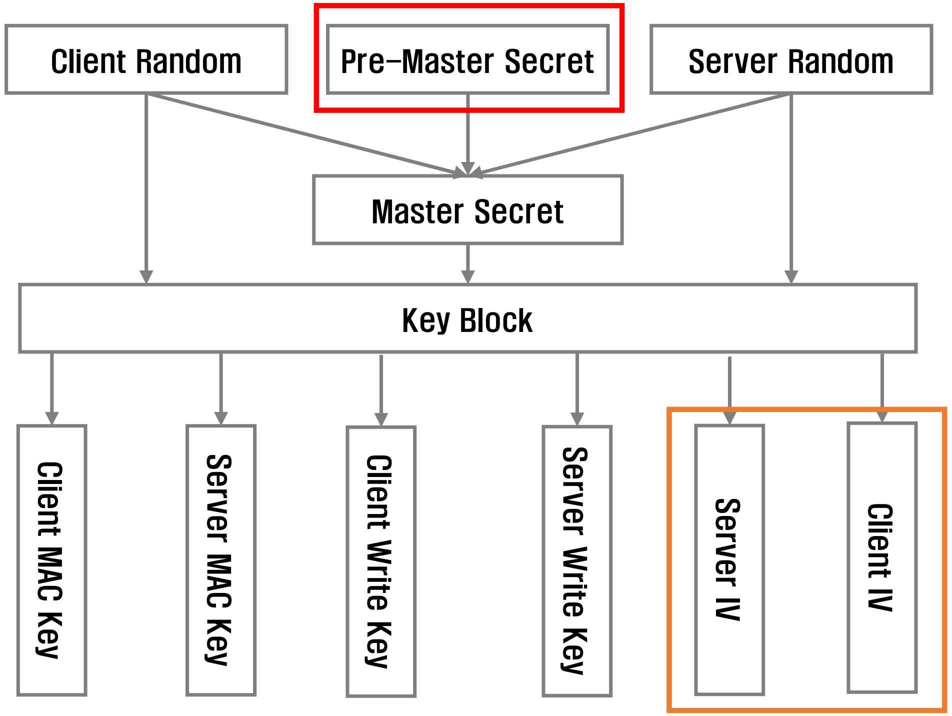
Only client and server know pre-master secret!

Master secret:

generated by both (client & server) using pre-master secret & nonces which were initially exchanged between client & server. 48 bytes long!

Session keys:

sequence of bytes (4/6 x 32) generated using master secret & nonces, which is then split into 4/6 separate keys: 2 MAC Keys, 2 Encryption Keys, and optionally 2 Initialization Vector (IV) Keys.

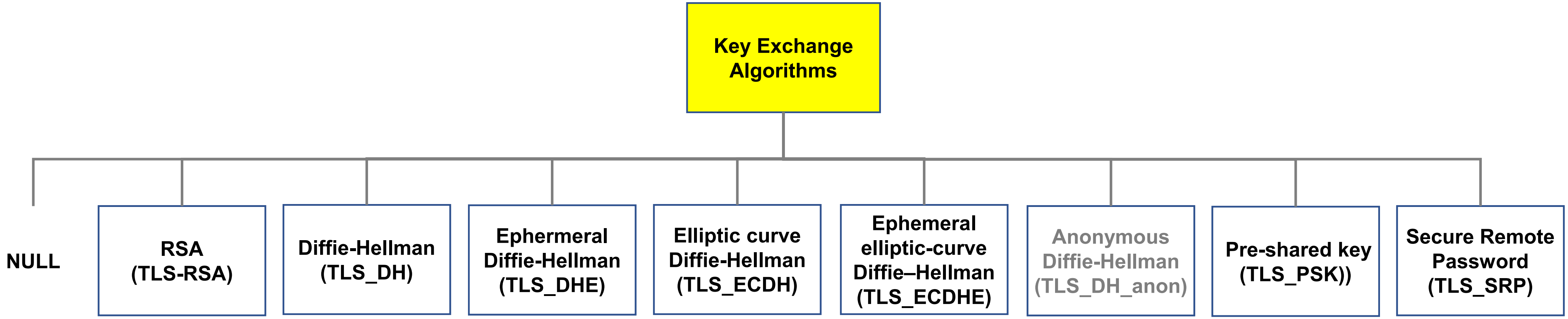


Optional, used only in case of some encryption algorithms!

Note : Methods for exchanging Pre-master Secrete

There are 8 possible approaches how pre-master key can be exchanged.

[자료, 2020] https://en.wikipedia.org/wiki/Transport_Layer_Security#Key_exchange_or_key_agreement



- TLS_DH_anon and TLS_ECDH_anon : Do not authenticate the server or the user and hence are rarely used because those are vulnerable to man-in-the-middle attacks. Only TLS_DHE and TLS_ECDHE provide forward secrecy.
- Public key certificates vary in the size of the public/private encryption keys used during the exchange and hence the robustness of the security provided. In July 2013, Google announced that it would no longer use 1024-bit public keys and would switch instead to 2048-bit keys to increase the security of the TLS encryption

Note : Methods for exchanging Pre-master Secrete

[자료, 2020] https://en.wikipedia.org/wiki/Transport_Layer_Security#Key_exchange_or_key_agreement

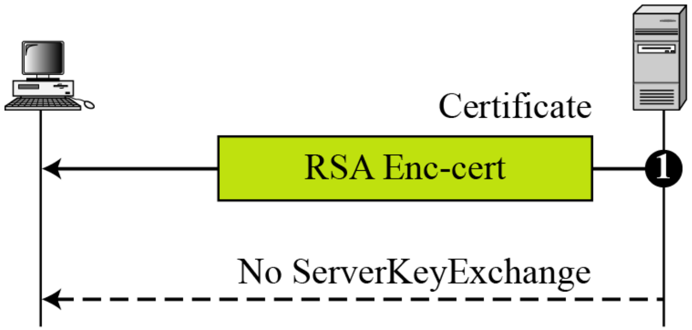
Key exchange/agreement and authentication

Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	Status
RSA	Yes	Yes	Yes	Yes	Yes	No	
DH-RSA	No	Yes	Yes	Yes	Yes	No	
DHE-RSA (forward secrecy)	No	Yes	Yes	Yes	Yes	Yes	
ECDH-RSA	No	No	Yes	Yes	Yes	No	
ECDHE-RSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
DH-DSS	No	Yes	Yes	Yes	Yes	No	
DHE-DSS (forward secrecy)	No	Yes	Yes	Yes	Yes	No[51]	
ECDH-ECDSA	No	No	Yes	Yes	Yes	No	
ECDHE-ECDSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
ECDH-EdDSA	No	No	Yes	Yes	Yes	No	
ECDHE-EdDSA (forward secrecy)[52]	No	No	Yes	Yes	Yes	Yes	Defined for TLS 1.2 in RFCs

Key exchange/agreement and authentication

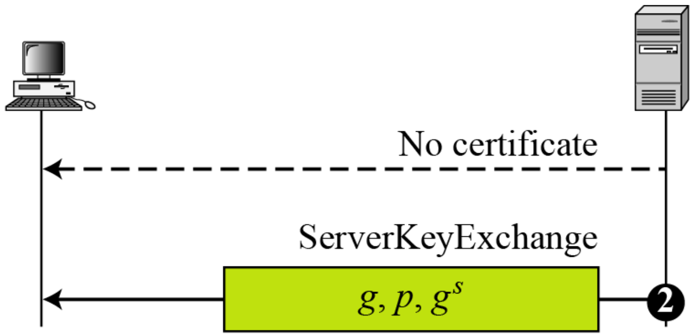
Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	Status
PSK	No	No	Yes	Yes	Yes		
PSK-RSA	No	No	Yes	Yes	Yes		
DHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
ECDHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
SRP	No	No	Yes	Yes	Yes		
SRP-DSS	No	No	Yes	Yes	Yes		
SRP-RSA	No	No	Yes	Yes	Yes		
Kerberos	No	No	Yes	Yes	Yes		
DH-ANON (insecure)	No	Yes	Yes	Yes	Yes		
ECDH-ANON (insecure)	No	No	Yes	Yes	Yes		
GOST R 34.10-94 / 34.10-2001[53]	No	No	Yes	Yes	Yes		Proposed in RFC drafts

Note : 4 cases of Phase 2 of TLS Handshake



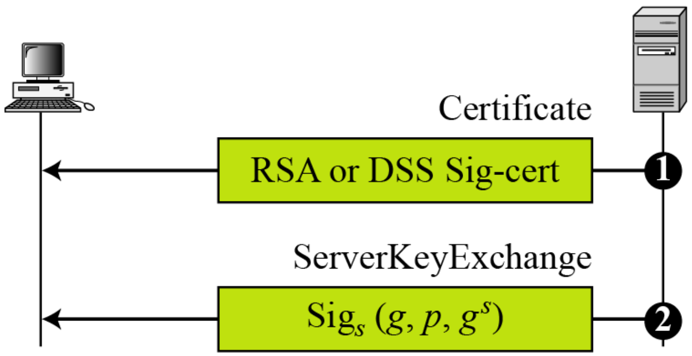
a. RSA

- Pre-master secret generated only by C, encrypted using S's public key and sent back to S in the next step



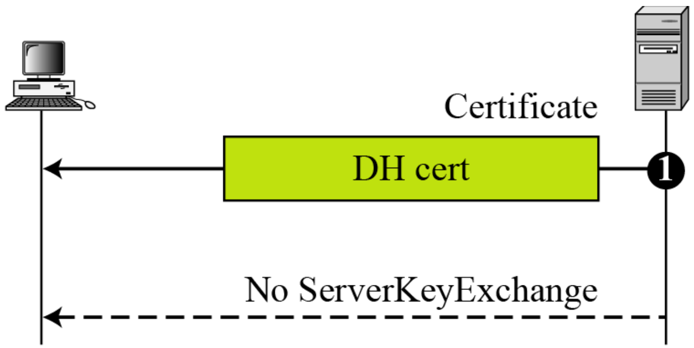
b. Anonymous DH

- Pre-master secret generated only jointly by C and S using DH



c. Ephemeral DH

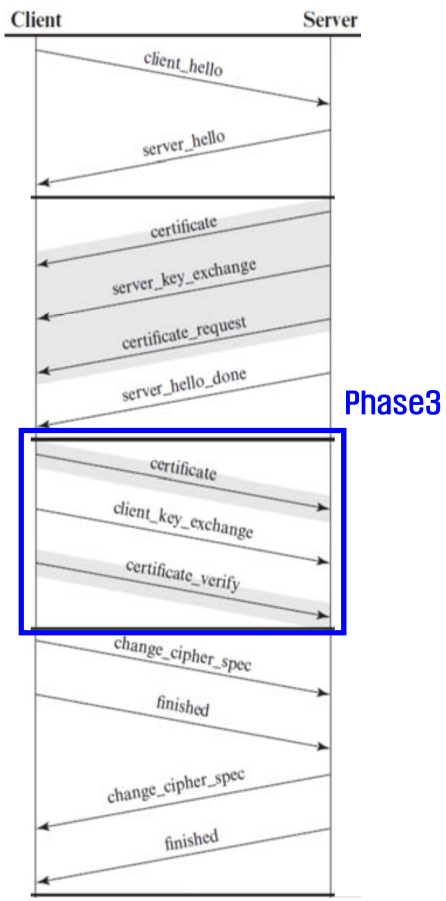
- Pre-master secret generated jointly by C & S using DH; DH half-keys encrypted when exchanged



d. Fixed DH

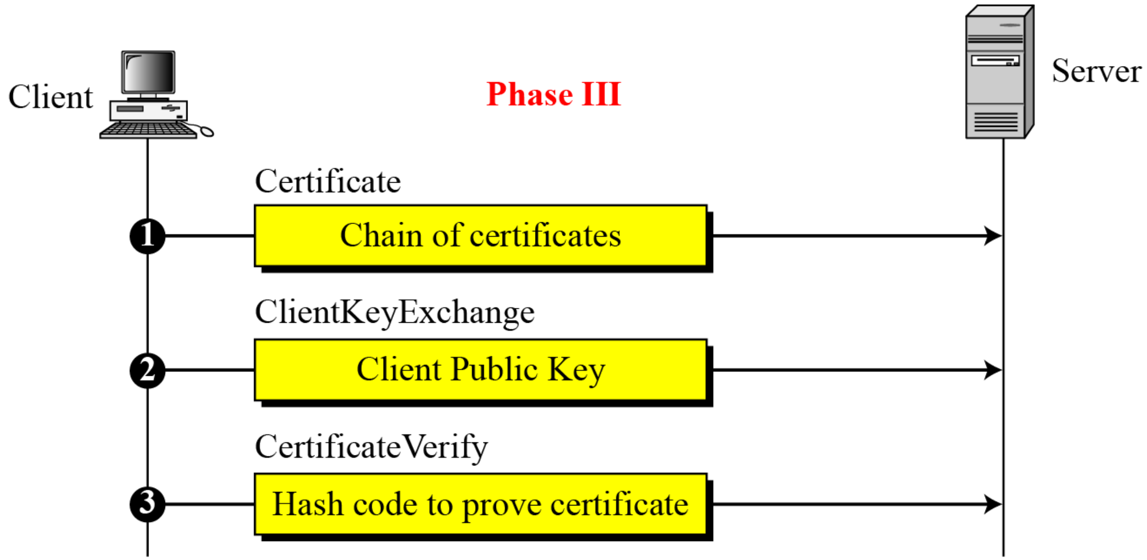
- Pre-master secret generated jointly by C & S using DH; DH half-keys placed in S’ s and C’ s certificates

Handshake Phase 3 – Client Key Exchange & Authentication



= client sends up to 3 messages back to the server – but only after it has verified that server provided a valid certificate & acceptable HelloServer parameters

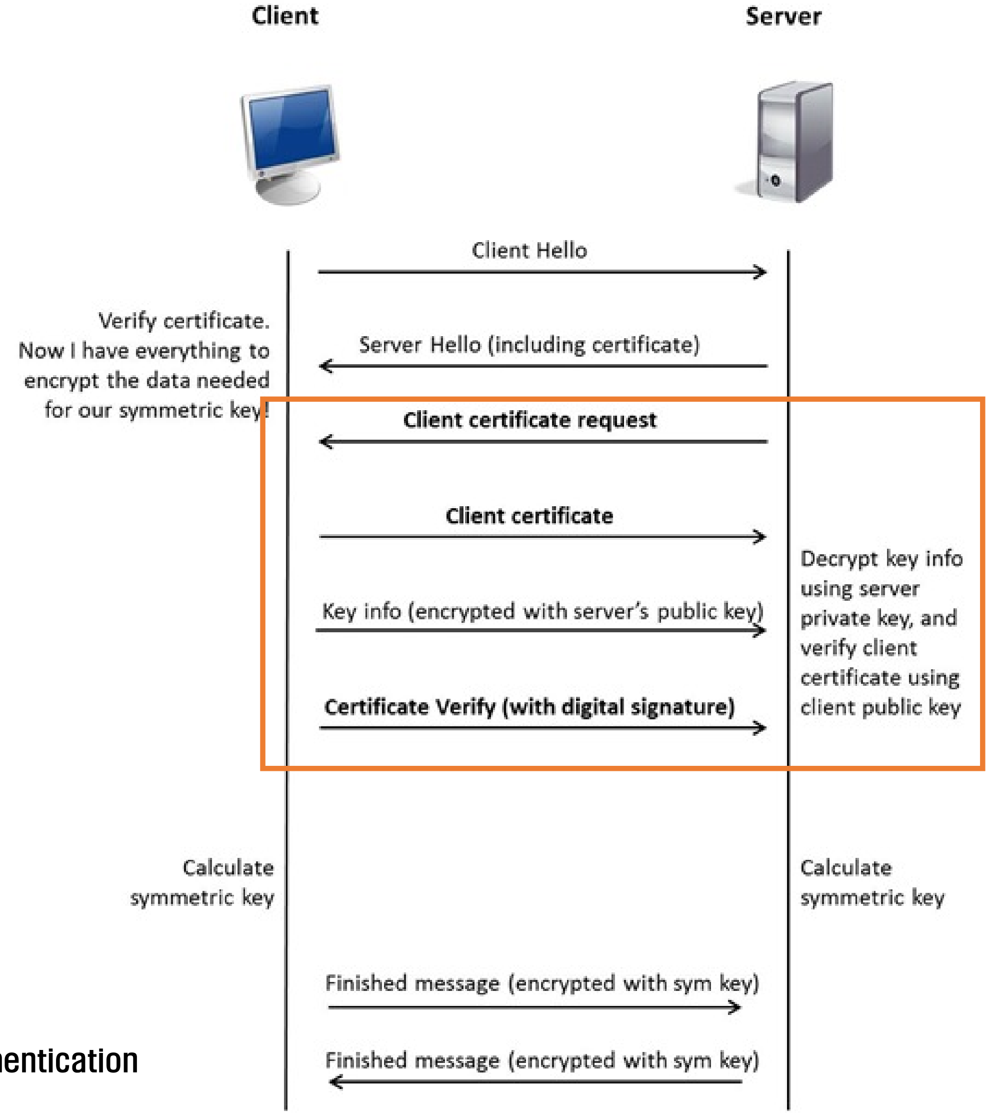
① **Certificates**= to certify itself to server, client sends a Certificate message – the format is the same in Phase II, but the content is different. This message optional – only sent if requested by server



Handshake Phase 3

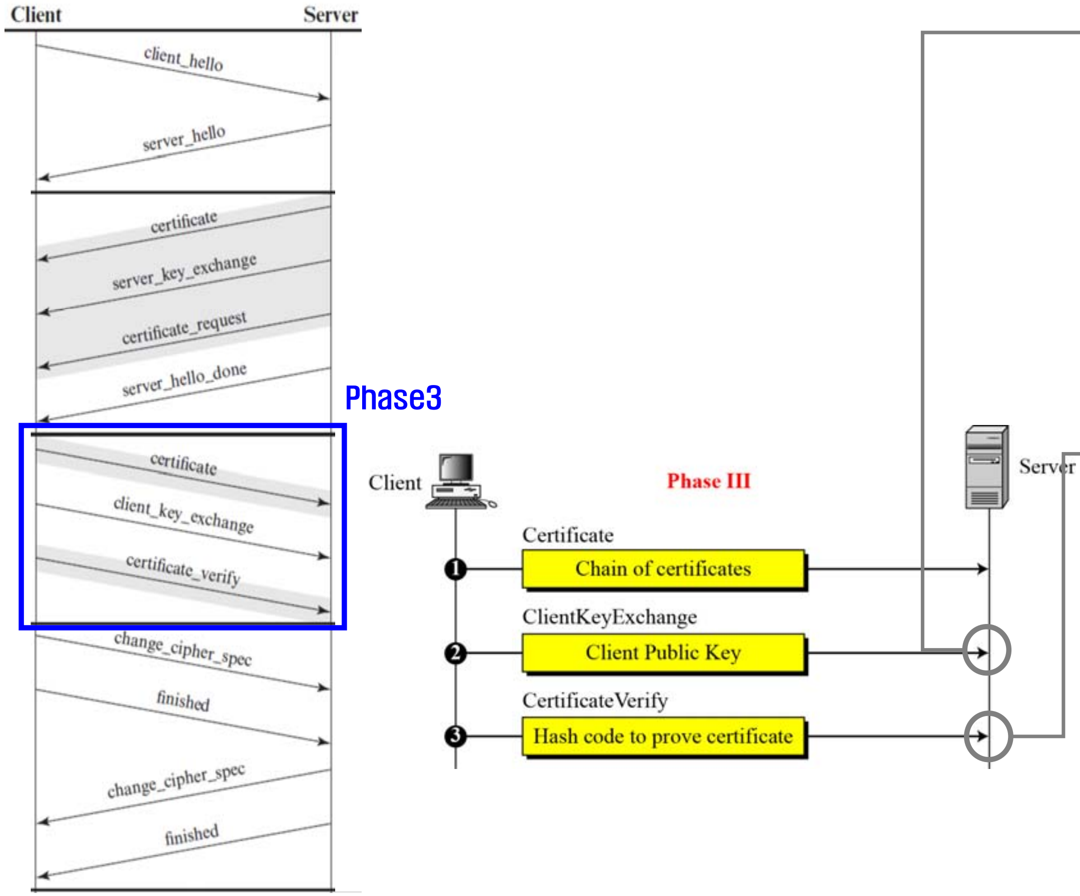
Client Authentication

- During this handshake, client authenticates server's identity by verifying server certificate. Although client always must authenticates server's identity, server is not required to authenticate client's identity. However, there are situations that call for server to authenticate client. Client authentication is a feature that lets you authenticate users that are accessing a server.
- Client authentication allow you to rest assured that the person represented by the certificate is the person you expect. Many companies want to ensure that only authorized users can gain access to the services and content they provide. As more personal and access-controlled information moves online, client authentication becomes more of a reality and a necessity.



출처 : https://devcentral.f5.com/s/articles/ssl-profiles-part-8-client-authentication

Handshake Phase 3 – Client Key Exchange & Authentication



② ClientKeyExchange= contains client's contribution to the pre-master secret. The contents of this message are based on the key-exchange algorithm used.

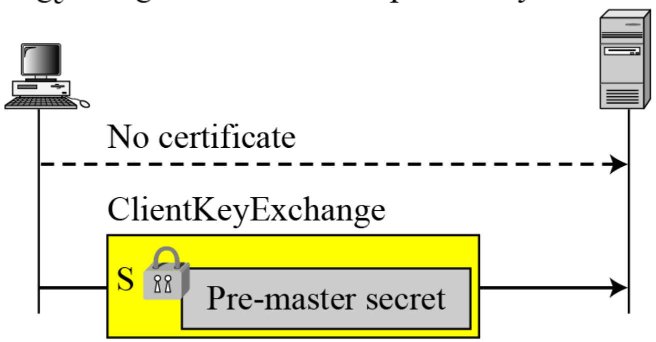
- RSA : client creates the entire pre-master secret and encrypts it with server' s RSA public key. If case of anonymous or ephemeral DH, client sends its DF half-key. In case of fixed DH, the contents of this message are empty …

③ CertificateVerfiy= if client has sent a certificate declaring that it owns the public key in the certificate, it needs to prove that it knows the corresponding private key (in order to thwart an impostor who sends the certificate and claims that it comes from client). The proof of private-key possession is done by creating a message and signing it with client' s private key …

Note : 4 cases of Phase 3 of TLS Handshake

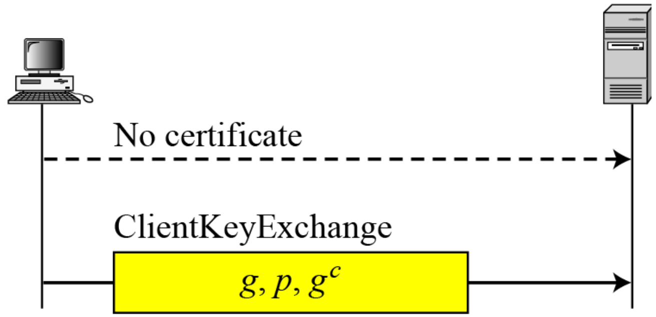
S crypted with server's public key

Sig_c: Signed with client's public key



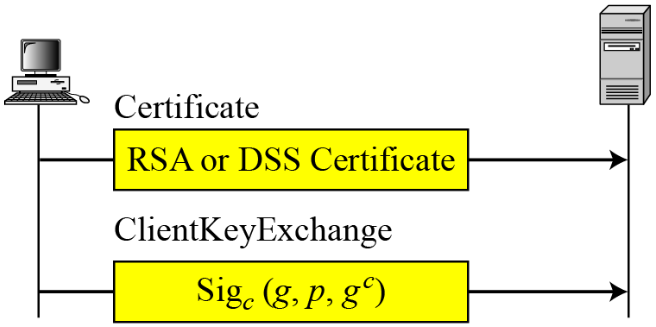
a. RSA

- No certificate unless the server has requested
- ClientKeyExchenage message includes the pre-master key encrypted with RSA public key received in Phase 2.



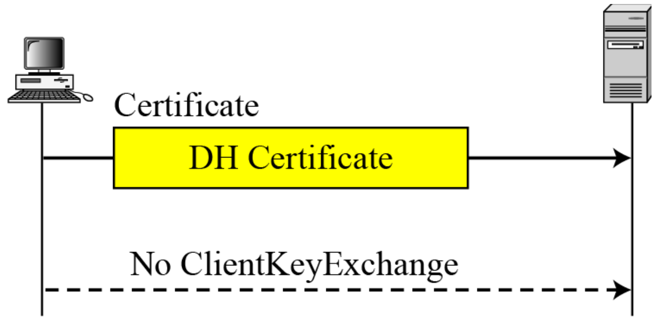
b. Anonymous DH

- no certificate because C and S are anonymous.
- In ClientKeyExchange message, S send DH parameters and its half-key. C is not authenticated to S.



c. Ephemeral DH

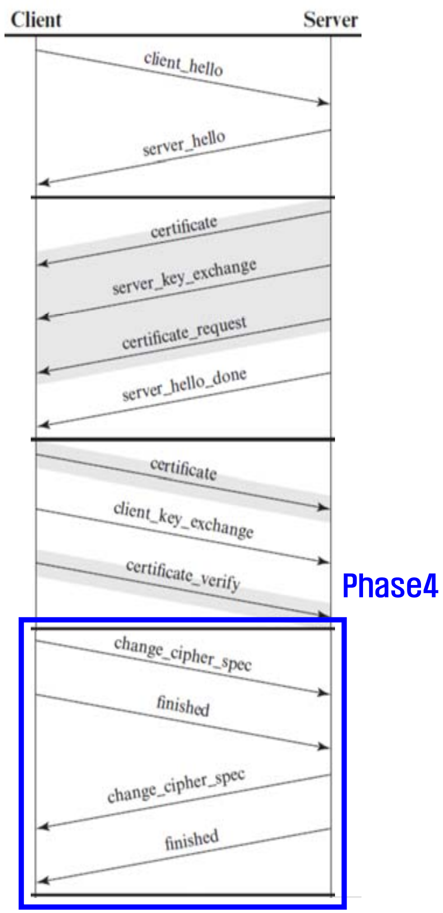
- C has a certificate
- In ClientKeyExchenage message, C signs DH parameters and its half-key and send them. C is authenticated to S by signing this message.



d. Fixed DH

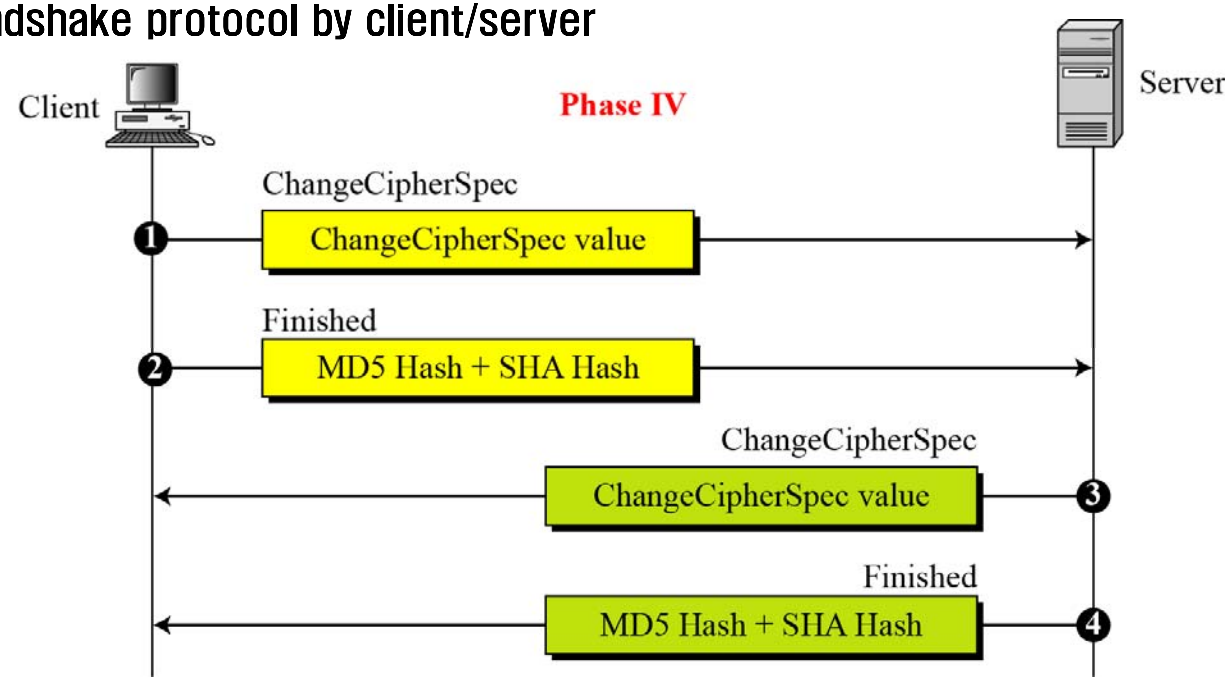
- C sends a DH certificate.
- No second message. C is authenticatd to S by sending DH certificates.

Handshake Phase 4 – Finalizing & Finishing



= Completes the setting up of a secure connection

- ✓ **ChangeCipherSpec**= message that is actually part of **ChangeCipherSpec protocol**– shows that client/server has moved all of the cipher suite set and parameters form pending to active state –encryption begins from this point
- ✓ **Finished**= encrypted and authenticated message that announces the end of handshake protocol by client/server



TLS Change Cipher Spec Protocol

- Simplest – consists of a 1 single byte with value 1.
  - ✓ The sole purpose of this message is to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection.
    - ❖ by sending this message, client & server tell each other: "*everything I tell you from now on will be authenticated & encrypted*" (if authentication/encryption was requested)
    - ❖ by successfully decrypting subsequent Finish messages & verifying their hash-es & MAC-s client & server ensure that connection & its parameters are successfully set; otherwise handshake is considered failed & connection should be torn down

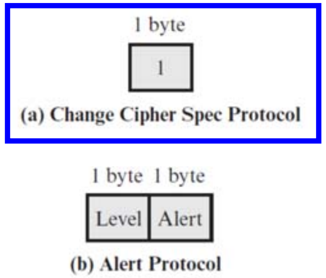


Figure 17.5 TLS Record Protocol Payload

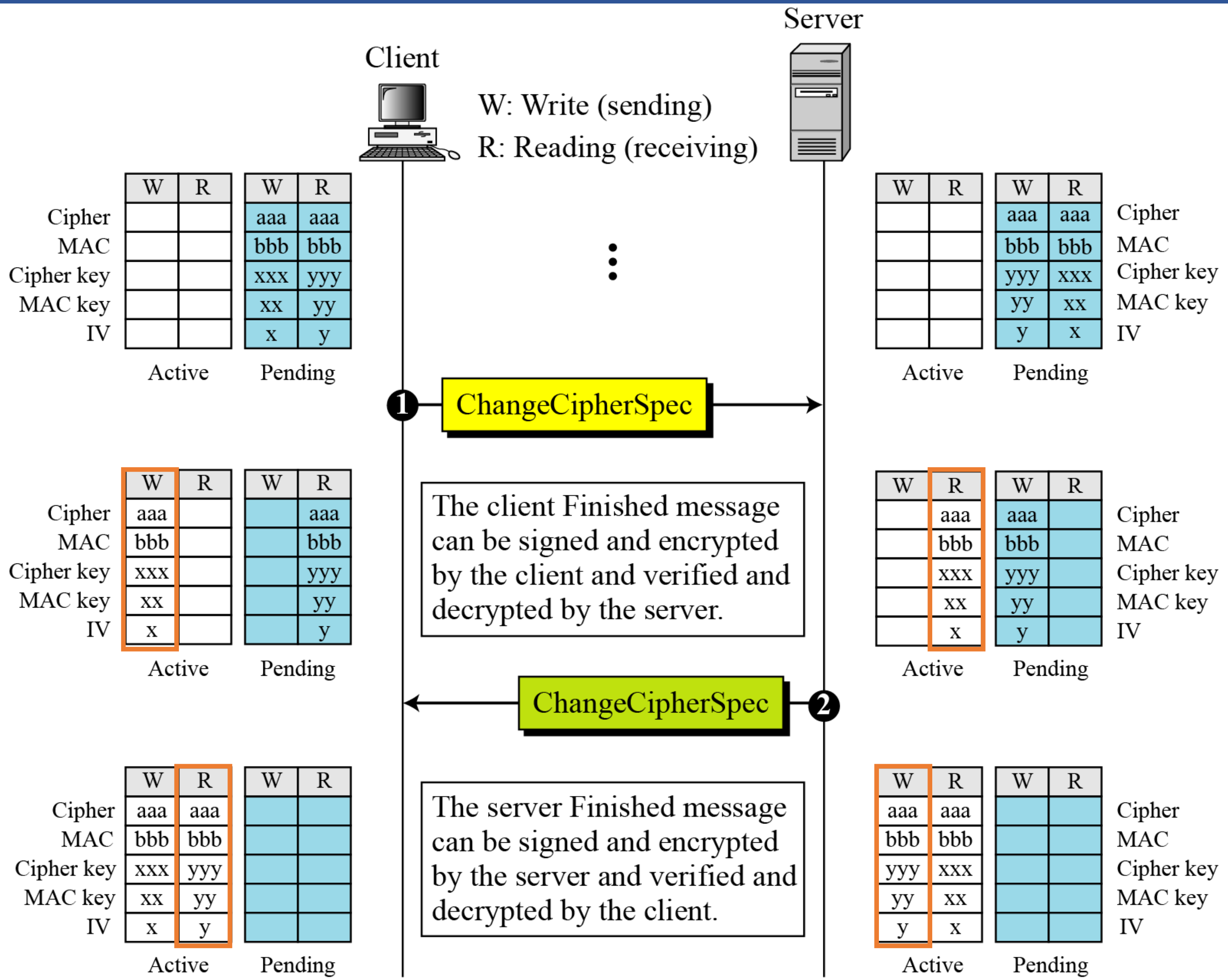
Hex	Dec	Type
0x14	20	ChangeCipherSpec
0x15	21	Alert
0x16	22	Handshake
0x17	23	Application
0x18	24	Heartbeat

Record Protocol Header

+	Byte +0	Byte +1	Byte +2	Byte +3
Byte 0	20			
Bytes 1..4	Version		Length	
	(Major)	(Minor)	0	1
Byte 5	CCS protocol type			

Currently only 1 !

Example **: Movement of TLS Parameters from pending to active state**



TLS Alert Protocol

- Used to convey TLS related alerts (warnings or errors) to peer entity – consists of 2 bytes only
  - ✓ similar to other message carried by TLS Record Protocol, alert messages are compressed and encrypted, as specified
  - ✓ level type = 1 (warning) means connection or security may be unstable –recipient may decide to close session
  - ✓ level type = 2(fatal) means connection or security may be compromised –sender closes the session immediately

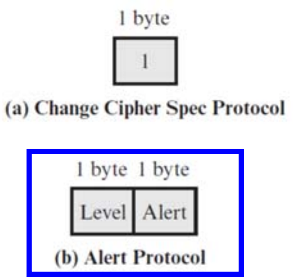


Figure 17.5 TLS Record Protocol Payload

Hex	Dec	Type
0x14	20	ChangeCipherSpec
0x15	21	Alert
0x16	22	Handshake
0x17	23	Application
0x18	24	Heartbeat

Record Protocol Header

+	Byte +0	Byte +1	Byte +2	Byte +3
Byte 0	21			
Bytes 1..4	Version		Length	
	(Major)	*(Minor)*	0	2
Bytes 5..6	Level	Description		
Bytes 7..(*p*−1)	MAC (optional)			
Bytes *p*..(*q*−1)	Padding (block ciphers only)			

2 bytes of data :
- ✓ Level identifies the level of alert
- ✓ Descritption identifies which type of alert is being sent

TLS Alert Protocol

- An alert signal includes a level indication which may be either fatal or warning (under TLS1.3 all alerts are fatal).
- Fatal alerts always terminate the current connection, and prevent future re-negotiations using the current session ID.

[출처] https://www.gnutls.org/manual/html_node/The-TLS-Alert-Protocol.html

Example : TLS Alert codes

- Alert description types (Wikipedia)

Code	Description	Level types	Note
0	Close notify	warning/fatal	
10	Unexpected message	fatal	
20	Bad record MAC	fatal	Possibly a bad SSL implementation, or payload has been tampered with e.g. FTP firewall rule on FTPS server.
21	Decryption failed	fatal	TLS only, reserved
22	Record overflow	fatal	TLS only
30	Decompression failure	fatal	
40	Handshake failure	fatal	
41	No certificate	warning/fatal	SSL 3.0 only, reserved
42	Bad certificate	warning/fatal	
43	Unsupported certificate	warning/fatal	e.g. certificate has only Server authentication usage enabled and is presented as a client certificate
44	Certificate revoked	warning/fatal	
45	Certificate expired	warning/fatal	Check server certificate expire also check no certificate in the chain presented has expired
46	Certificate unknown	warning/fatal	
47	Illegal parameter	fatal	
48	Unknown CA (Certificate authority)	fatal	TLS only
49	Access denied	fatal	TLS only – e.g. no client certificate has been presented (TLS: Blank certificate message or SSLv3: No Certificate alert), but server is configured to require one.
50	Decode error	fatal	TLS only
51	Decrypt error	warning/fatal	TLS only
60	Export restriction	fatal	TLS only, reserved
70	Protocol version	fatal	TLS only
71	Insufficient security	fatal	TLS only
80	Internal error	fatal	TLS only
86	Inappropriate Fallback	fatal	TLS only
90	User canceled	fatal	TLS only
100	No renegotiation	warning	TLS only
110	Unsupported extension	warning	TLS only
111	Certificate unobtainable	warning	TLS only
112	Unrecognized name	warning/fatal	TLS only; client's Server Name Indicator specified a hostname not supported by the server
113	Bad certificate status response	fatal	TLS only

TLS Vulnerabilities

As of August 2019, Trustworthy Internet Movement estimate the ratio of websites that are vulnerable to TLS attacks.

Survey of the TLS vulnerabilities of the most popular websites

Attacks	Security			
	Insecure	Depends	Secure	Other
Renegotiation attack	0.3% support insecure renegotiation	0.1% support both	98.4% support secure renegotiation	1.1% no support
RC4 attacks	1.2% support RC4 suites used with modern browsers	12.1% support some RC4 suites	86.7% no support	N/A
TLS Compression (CRIME attack)	0.6% vulnerable	N/A	N/A	N/A
Heartbleed	<0.1% vulnerable	N/A	N/A	N/A

Survey of the TLS vulnerabilities of the most popular websites

Attacks	Security			
	Insecure	Depends	Secure	Other
ChangeCipherSpec injection attack	0.2% vulnerable and exploitable	1.2% vulnerable, not exploitable	96.9% not vulnerable	1.7% unknown
POODLE attack against TLS (Original POODLE against SSL 3.0 is not included)	0.3% vulnerable and exploitable	N/A	99.5% not vulnerable	0.2% unknown
Protocol downgrade	11.3% Downgrade defence not supported	N/A	71.6% Downgrade defence supported	17.0% unknown

TLS v1.3

Objectives

- *Clean up* : Remove unused or unsafe features
- *Security* : Improve security by using modern security analysis techniques
- *Privacy* : Encrypt more of the protocol
- *Performance* : Our target is a 1-RTT handshake for naive clients; 0-RTT handshake for repeat connections
- *Continuity* : Maintain existing important use cases

Removed Features

- *Static RSA*
- *Custom (EC)DHE groups*
- *Compression*
- *Renegotiation*
- *Non-AEAD ciphers*
- *Simplified resumption*

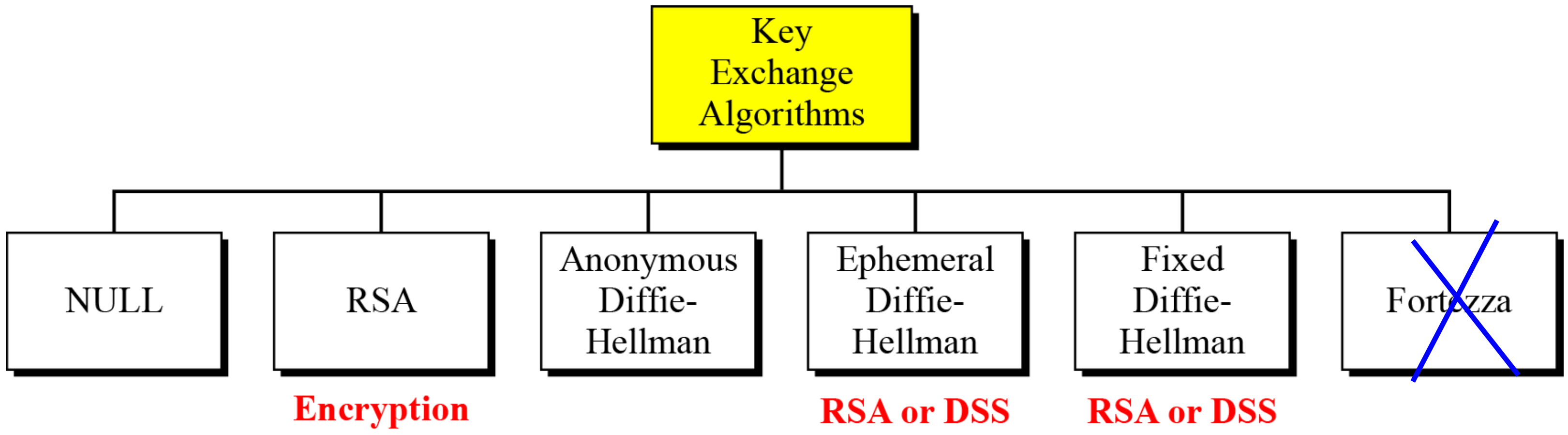
출처 : Stanford Univ., ee380, TLS1.3, Lecture Note

부 록

SSL/TLS 부록

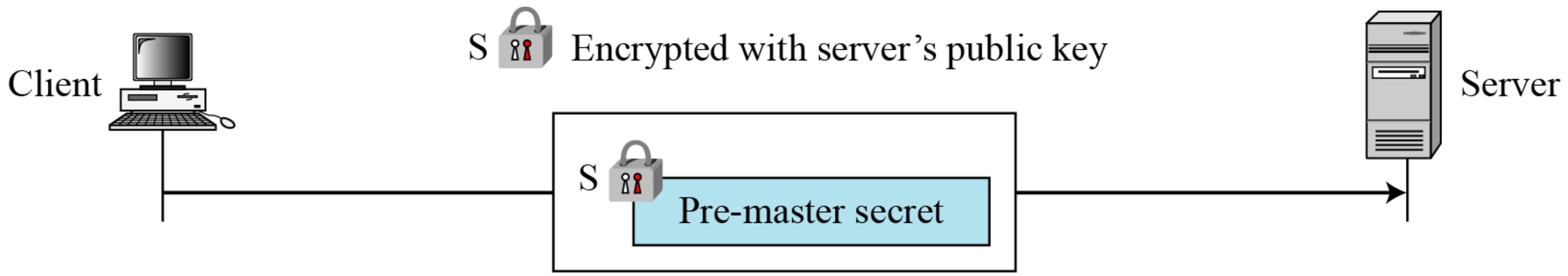
SSL / TLS – 키 교환

- 키 교환 방법

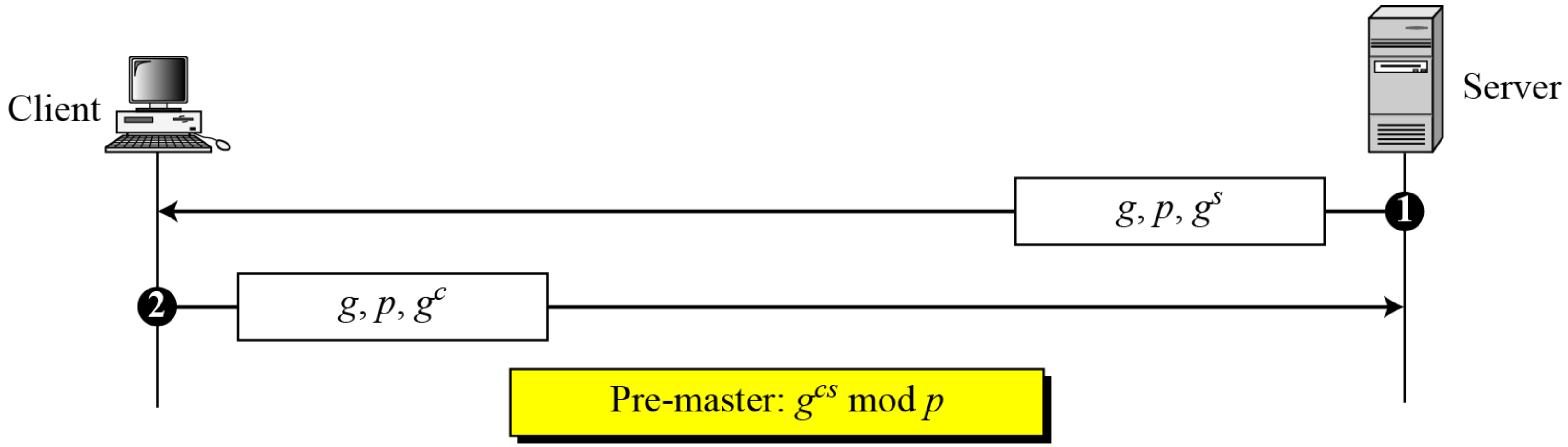


- 클라이언트와 서버는 사전-마스터 비밀 값을 알 필요가 있다.

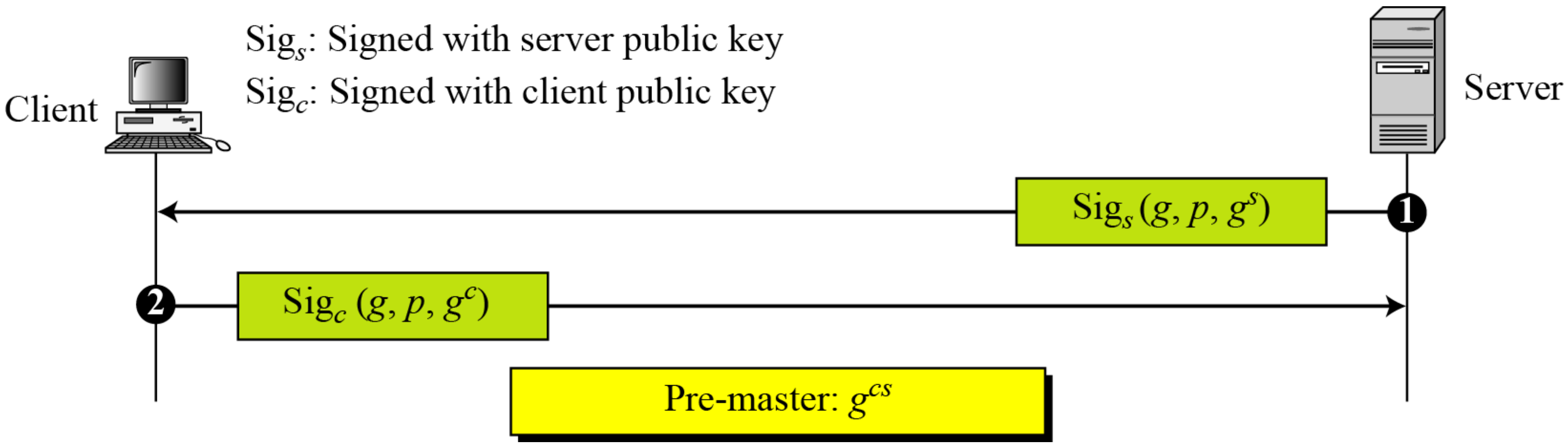
- **RSA 키 교환: 서버 공개키**



- **Anonymous Diffie-Hellman 키 교환**



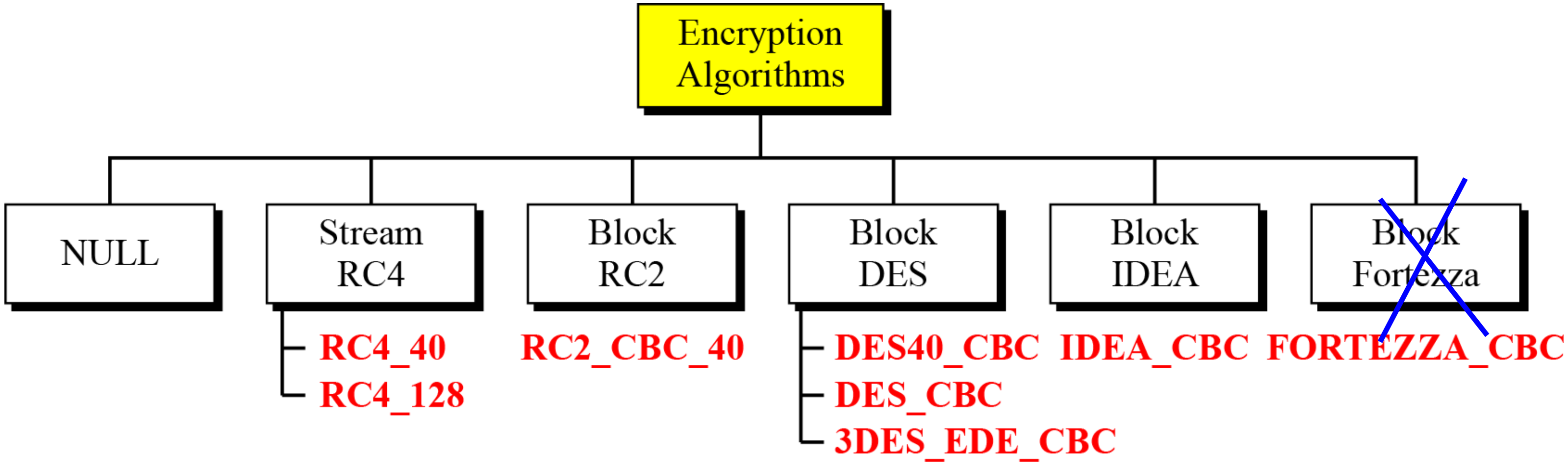
- **임시 Diffie-Hellman 키 교환**



- **고정(Fixed) Diffie-Hellman**

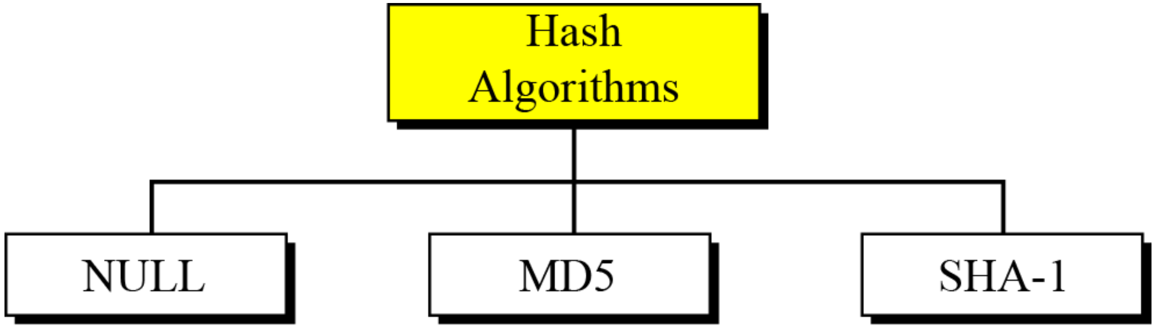
또 다른 해는 고정 Diffie-Hellman 방법이다. 그룹에 있는 모든 개체는 고정 Diffie-Hellman 파라메터(g and p)를 준비할 수 있다.

- 암/복호화 알고리즘



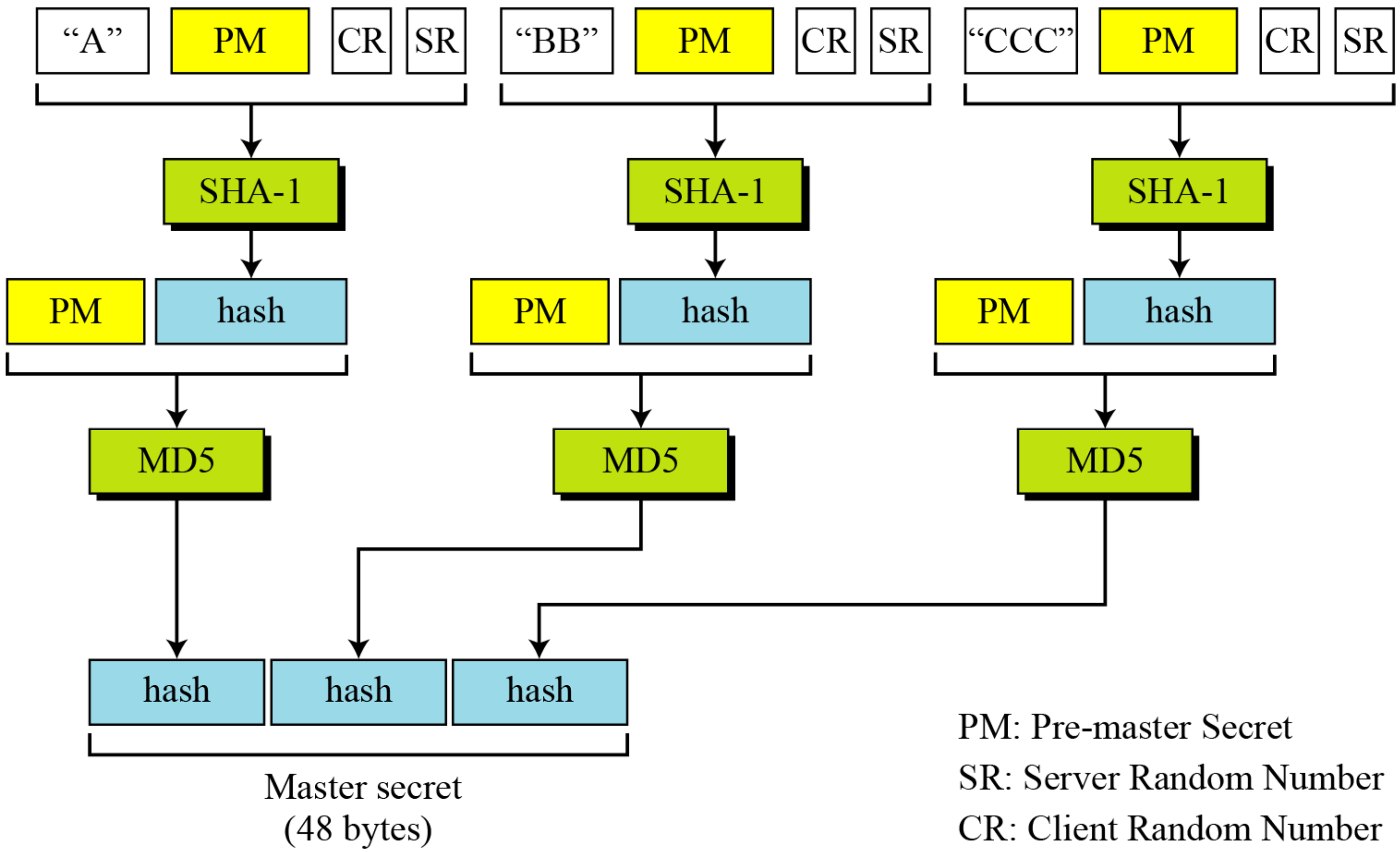
- NULL : The NULL category simply defines the lack of an encryption/decryption algorithm.
- Stream RC : Two RC algorithms are defined in stream mode.
- Block RC : One RC algorithm is defined in block mode.
- DES : All DES algorithms are defined in block mode.
- IDEA : 블록 모드에서 규정된 IDEA 알고리즘은 128비트 키를 갖는 IDEA_CBC 이다

- 메시지 무결성을 위한 해쉬 알고리즘



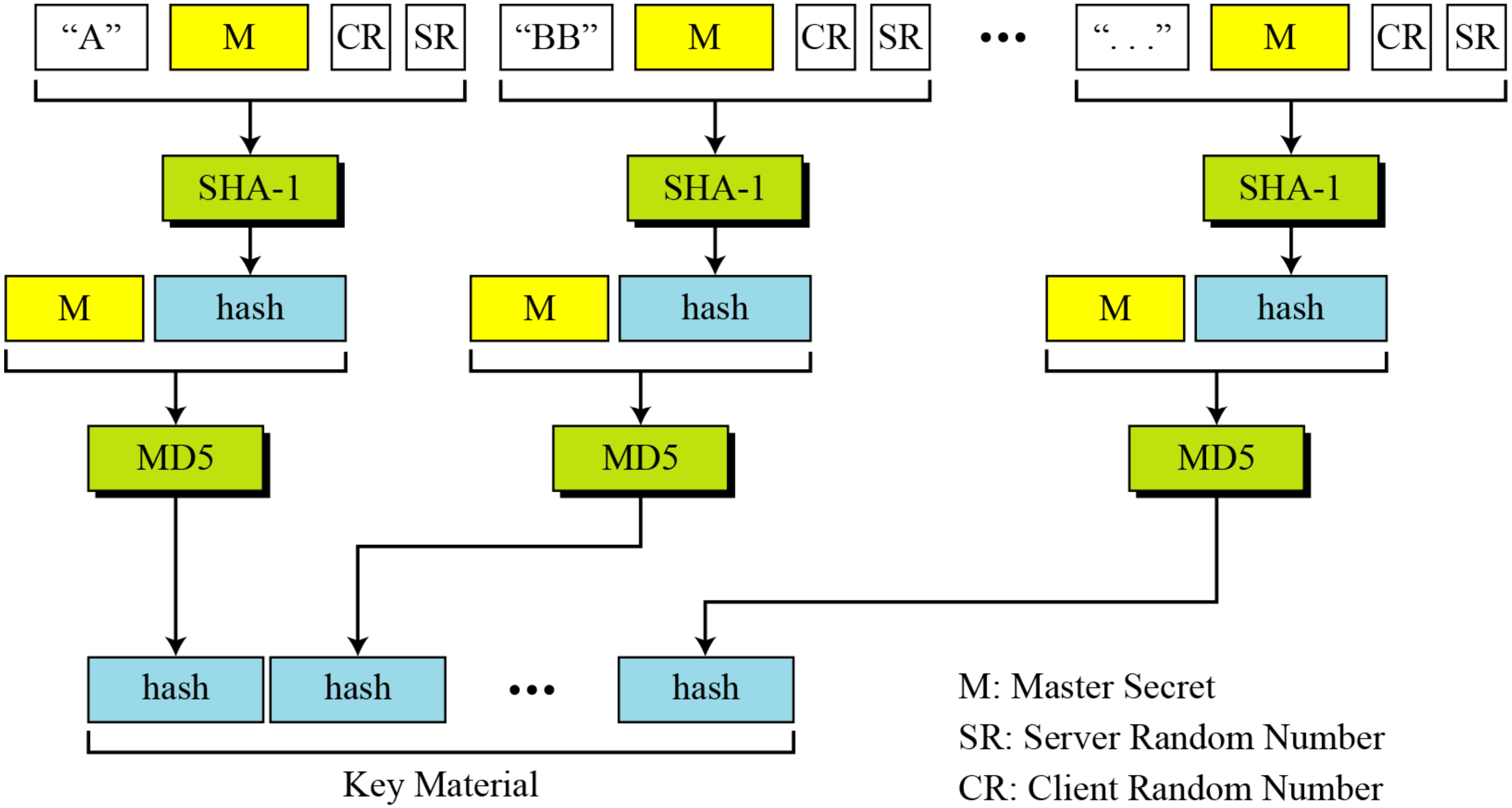
- NULL : 두 당사자가 알고리즘의 사용을 선언할 수 있다. 이 경우에, 해쉬 함수는 없고 메시지는 인증되지 않는다.
- MD5 : 두 당사자가 해쉬 알고리즘으로 MD5 알고리즘을 선택할 수 있다. 이 경우에, 128비트-키 MD5 해쉬 알고리즘이 사용된다.
- SHA-1 : 두 당사자가 해쉬 알고리즘으로 SHA를 선택할 수 있다. 이 경우에, 160-비트 SHA-1 해쉬 알고리즘이 사용된다.

- **사전-마스터 비밀에서 마스터 비밀의 계산**

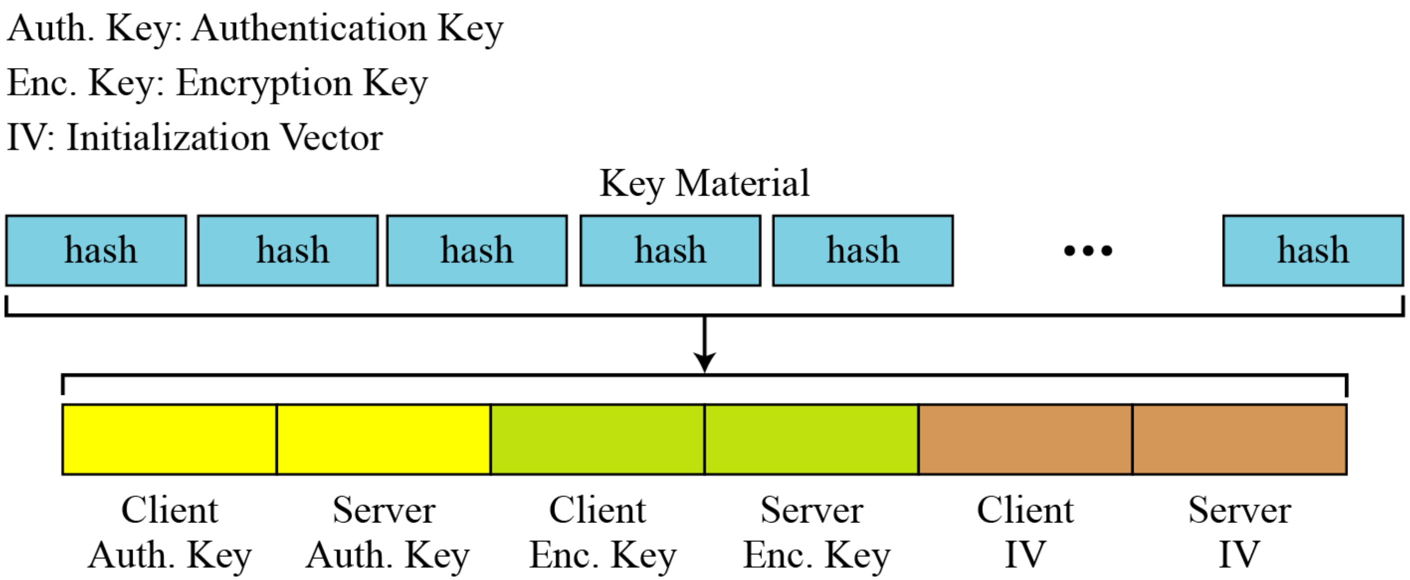


SSL / TLS – Cryptographic Parameter Generation

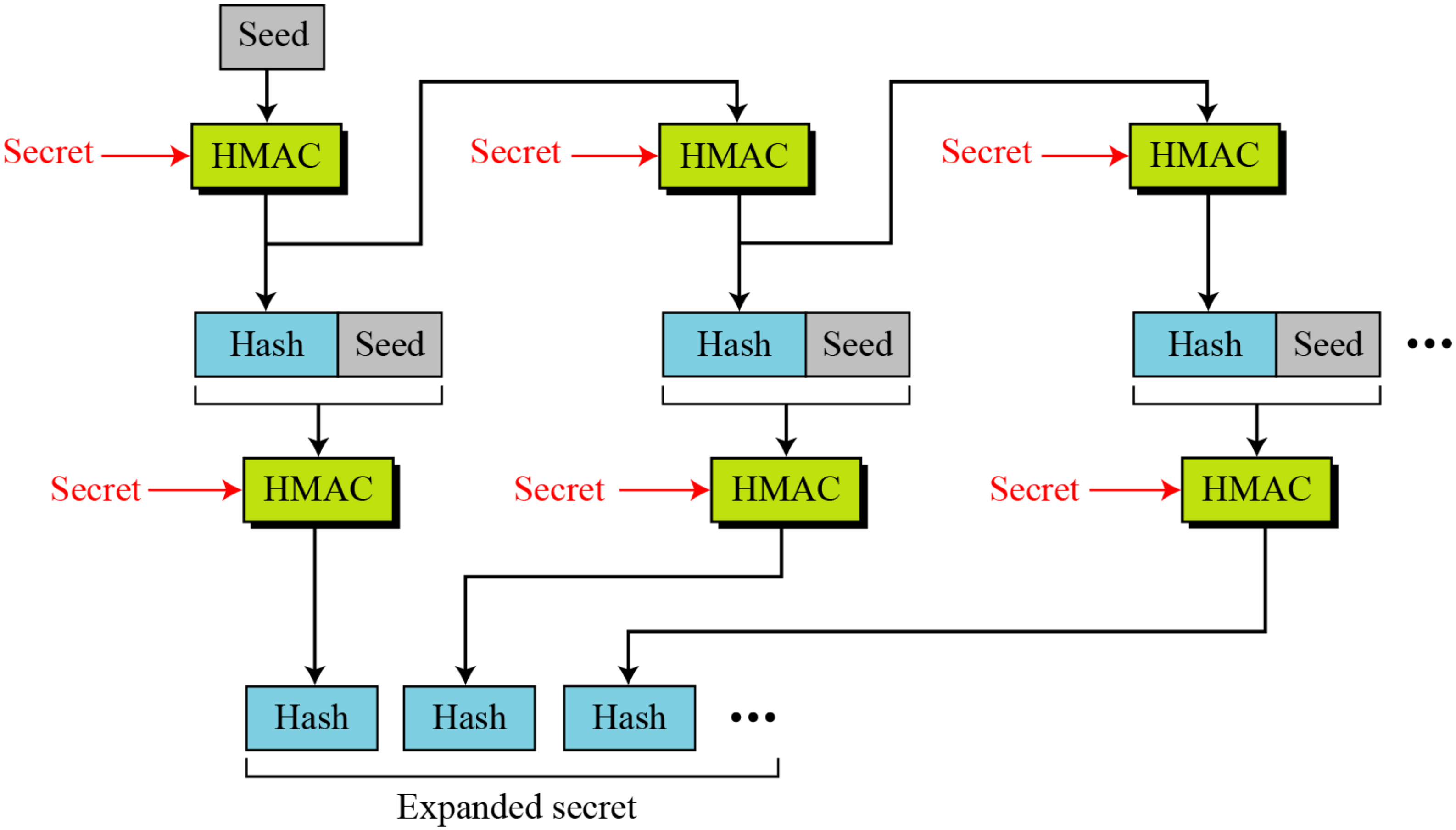
- **마스터 비밀로부터 키 재료 계산**



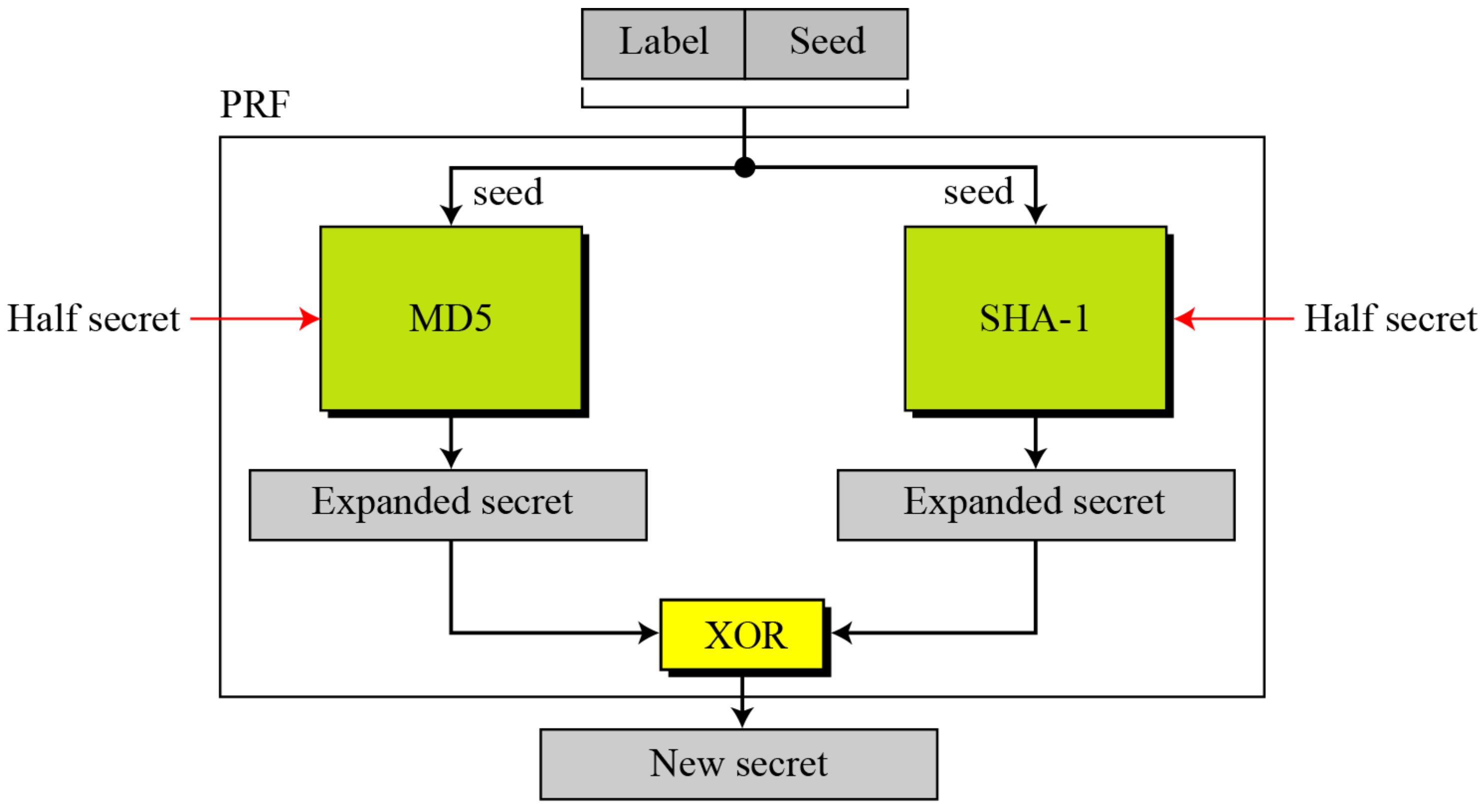
- 키 재료로부터 암호학적 비밀 추출



- 데이터 – 확장 함수 (TLS)

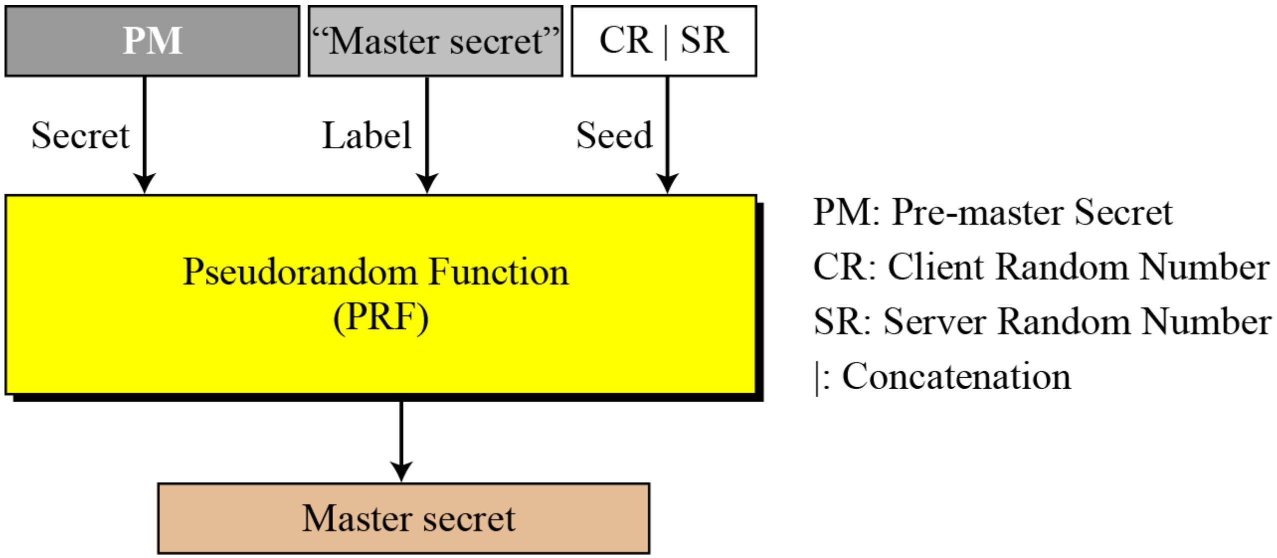

Seed
Secret
HMAC
Secret
HMAC
Secret
HMAC
Hash
Seed
Hash
Seed
Hash
Seed
•••
Secret
HMAC
Secret
HMAC
Secret
HMAC
Hash
Hash
Hash
•••
Expanded secret

- **PRF 함수 (TLS)**

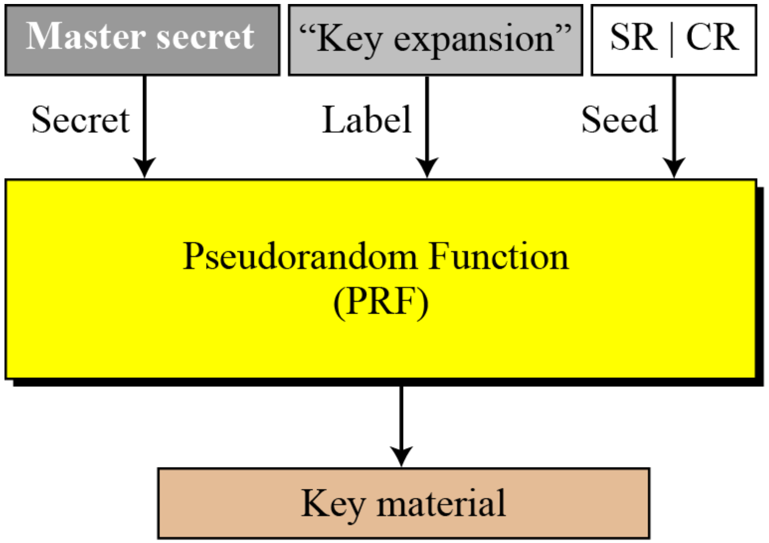


Label
Seed
PRF
seed
seed
Half secret
MD5
SHA-1
Half secret
Expanded secret
Expanded secret
XOR
New secret

- **마스터 비밀 생성 (TLS)**


PM
"Master secret"
CR | SR
Secret
Label
Seed
Pseudorandom Function (PRF)
Master secret
PM: Pre-master Secret
CR: Client Random Number
SR: Server Random Number
|: Concatenation

- 키 재료 생성 (TLS)



CR: Client Random Number
SR: Server Random Number
|: Concatenation

SSL.TLS Handshake
Wireshark Screenshot

출처 : https://www.linuxbabe.com/security/ssltls-handshake-process-explained-with-wireshark-screenshot

SSL/TLS Handshake Explained with Wireshark Screenshot

Step 1. Client Hello

- The client begins the communication.
- The first step is called client hello.
- The client lists the versions of SSL/TLS and cipher suites it' s able to use.

```
- Secure Sockets Layer
 ▾TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 186
  ▾Handshake Protocol: Client Hello
     Handshake Type: Client Hello (1)
     Length: 182
     Version: TLS 1.2 (0x0303)
    ▸Random
     Session ID Length: 0
     Cipher Suites Length: 22
    ▾Cipher Suites (11 suites)
       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_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_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
       Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
       Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
       Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
       Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
     Compression Methods Length: 1
```

SSL/TLS versions supported by client

Cipher Suites supported by client

Step 2. Server Hello

- The server will see the list of SSL/TLS versions and cipher suites and pick the newest the server is able to use.
- Then the server send a message to the client containing the SSL/TLS version and cipher suite it chose.

```
- Secure Sockets Layer
 ▾TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 70
  ▾Handshake Protocol: Server Hello
     Handshake Type: Server Hello (2)
     Length: 66
     Version: TLS 1.2 (0x0303)
    ▸Random
     Session ID Length: 0
     Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
     Compression Method: null (0)
     Extensions Length: 26
    ▸Extension: server_name
    ▸Extension: renegotiation_info
    ▸Extension: ec_point_formats
    ▸Extension: Application Layer Protocol Negotiation
```

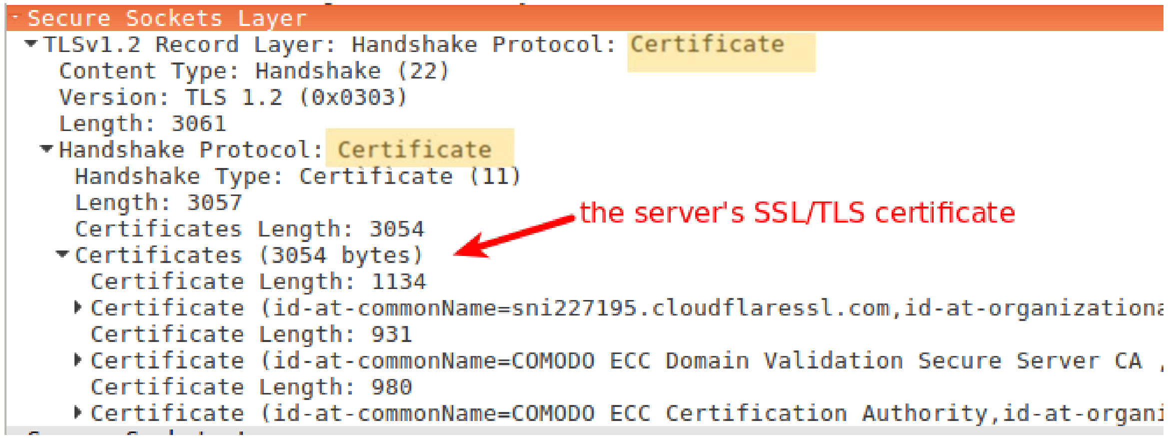
SSL/TLS version and cipher suite picked by the server

Step 3. Server Key Exchange

- After the server and client agrees on the SSL/TLS version and cipher suite, then server sends two things;

1) The first is its SSL/TLS certificate to the client.

- ✓ The client (web browser) validates the server' s certificate. Web browsers store a list of Root CA(Certificate Authority) in itself. These root CAs are third parties that are trusted by web browsers. The server' s certificate is issued by root CA or immediate CA. Immediate CA is a CA that is trusted by root CA.
- ✓ Web browsers trust Root CA. Root CA trust immediate CA. If the server' s certificate is issued by a trusted root CA or immediate CA, then the browser trust the server' s certificate. I will tell you how to find these root CAs in your web browser at the end of this article.



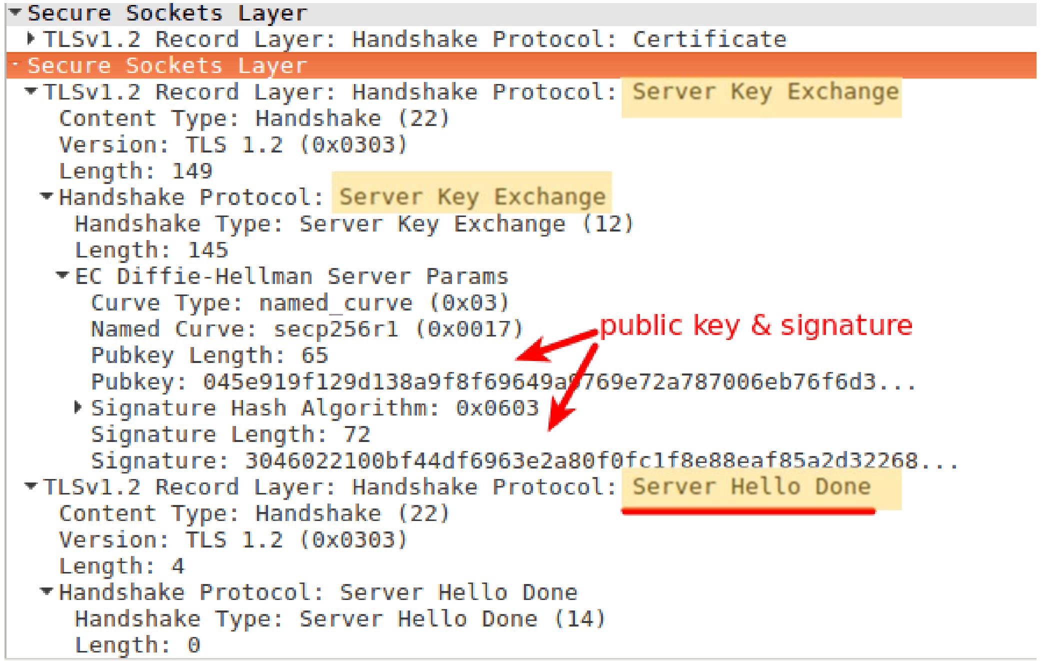
SSL./TLS Handshake Explained with Wireshark Screenshot

Step 3. Server Key Exchange

- After the server and client agrees on the SSL/TLS version and cipher suite, then server sends two things;

2) The second thing the server sends is its public key and signature.

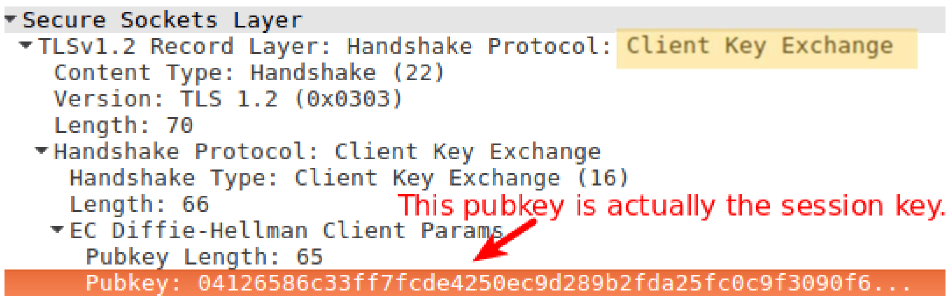
- ✓ The public key is actually included in the certificate. The client and the server encrypt message with the public key and it can only be decrypted with the private key. The server never share its private key with anyone.



```
▾ Secure Sockets Layer
  ▸ TLSv1.2 Record Layer: Handshake Protocol: Certificate
▾ Secure Sockets Layer
  ▾ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 149
    ▾ Handshake Protocol: Server Key Exchange
        Handshake Type: Server Key Exchange (12)
        Length: 145
      ▾ EC Diffie-Hellman Server Params
          Curve Type: named_curve (0x03)
          Named Curve: secp256r1 (0x0017)
          Pubkey Length: 65
          Pubkey: 045e919f129d138a9f8f69649a?769e72a787006eb76f6d3...
        ▸ Signature Hash Algorithm: 0x0603
          Signature Length: 72
          Signature: 3046022100bf44df6963e2a80f0fc1f8e88eaf85a2d32268...
  ▾ TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 4
    ▾ Handshake Protocol: Server Hello Done
        Handshake Type: Server Hello Done (14)
        Length: 0
```

Step 4. Client Key Exchange

- Until now, all the information sent between the client and server is unencrypted.
- Now the client receives the server's public key and generate a new session key (aka *pre-master key*) encrypted with the public key and sends it to the server.
  - ✓ The session key can only be decrypted with the private key and because only the server has the private key so only the client and server know the session key.
  - ✓ This session key is only valid in one session. If the user close the client and visit the same server next day, a new session key will be generated by the client.



Step 5. Change Cipher Spec

- The change cipher spec message is sent by both the client and server to notify the receiving party that subsequent records will be protected under the just-negotiated CipherSpec and keys.

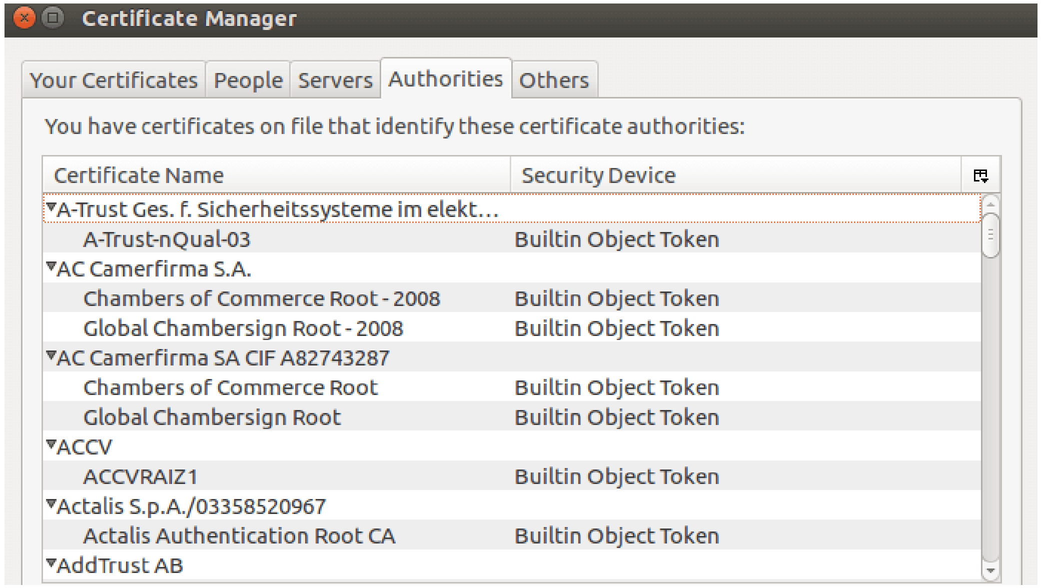
Step 6. Encrypted Handshake

- The client and the server sends to each other an encrypted message saying the key information is correct.
- Now the client (web browser) will see a green lock in the address bar. The client and server encrypt http traffic with the session key.

How to View Root CAs in Browser

Firefox

- Go to Tools › Options › Advanced › Certificate › View Certificate.



Chrome

- Go to settings › show advanced settings › manage certificate › authorities.