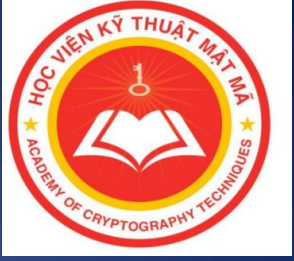
HỌC VIỆN KỸ THUẬT MẬT MÃ
KHOA AN TOÀN THÔNG TIN
****



GIAO THỨC AN TOÀN MẠNG

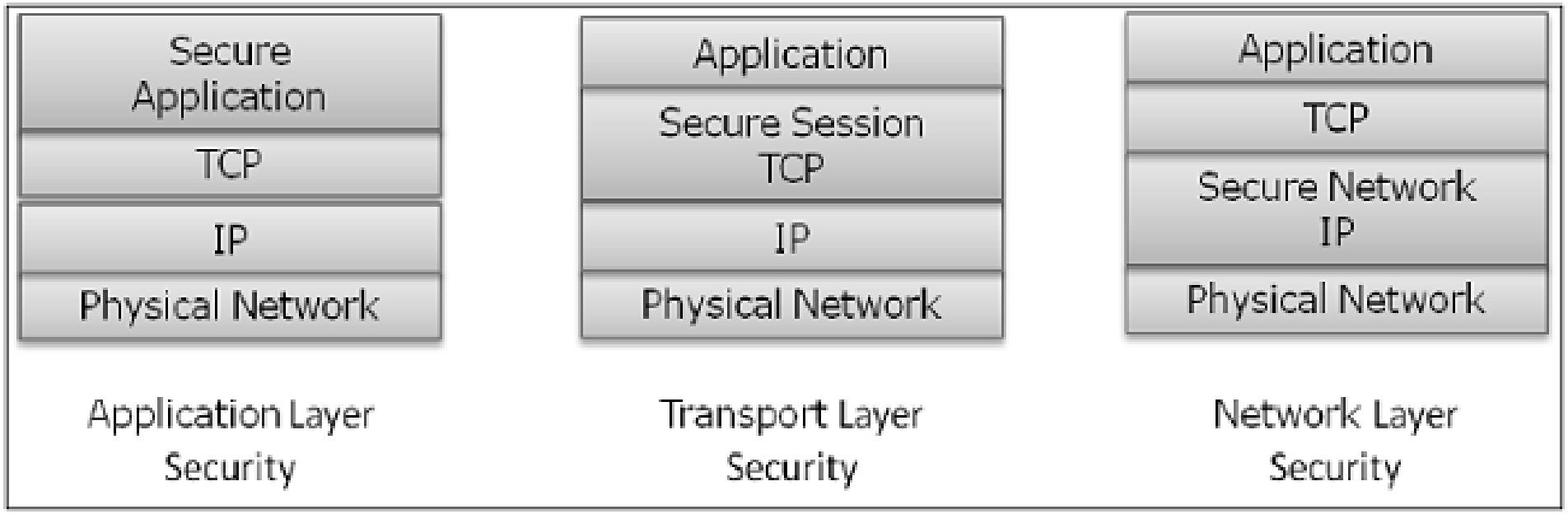
Bài 3.2. Bảo vệ tầng ứng dụng

TS. Trần Thị Lượng

Giao thức tầng ứng dụng

- Host Configuration: DHCP
- Domain Name System: DNS
- Remote Login: TELNET, SSH
- File Transfer: FTP, TFTP
- World Wide Web: HTTP
- Electronic Mail: SMTP, POP, IMAP, MIME
- Network Management: SNMP
- ...

Phân lớp cơ chế an toàn



1. Bộ giao thức SSL/TLS
2. Bộ giao thức SSH

Mục tiêu bài học

❑Kiến thức

- Hiểu được cơ chế bảo vệ tầng ứng dụng bằng các giao thức an toàn ở tầng thấp hơn
- Nắm bắt được cơ chế hoạt động của một số bộ giao thức an toàn ở tầng giao vận

❑Kỹ năng

- Phân tích hoạt động của giao thức qua việc chặn thu lưu lượng mạng

Tài liệu tham khảo

1. Giáo trình "Giao thức an toàn mạng máy tính"// Chương 4 "**Các giao thức bảo mật dịch vụ**"
2. Behrouz A. Forouzan, "TCP/IP Protocol Suite" (4e)// Part 4 "**Application Layer**", Mc Graw Hill, 2010
3. André Perez, "Network Security"//Chapter 6.2 "**SSH Protocol**", Wiley, 2014
4. William Stallings, "**Protocol Basics: Secure Shell Protocol**"//The Internet Protocol Journal, Volume 12, No.4
   https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-46/124-ssh.html
5. **What is Network Security Transport Layer**?
   https://www.wisdomjobs.com/e-university/network-security-tutorial-449/network-security-transport-layer-21971.html

1. Bộ giao thức SSL/TLS
2. Bộ giao thức SSH

1. Tổng quan về SSL/TLS
2. Giao thức Record
3. Giao thức Handshake

1 Tổng quan về SSL/TLS

2 Giao thức Record

3 Giao thức Handshake

Tài liệu tham khảo bổ sung

1. Muhammad Rizwan Asghar, **Network Defence and Countermeasures**, Lectures 8 & 9
   [https://www.cs.auckland.ac.nz/courses/compsci726s2c/lectures/ra-2017/]
2. SSL Socket Communication
   [https://sites.google.com/site/ddmwsst/create-your-own-certificate-and-ca/ssl-socket-communication]
3. Transport Layer Security
   [https://en.wikipedia.org/wiki/Transport_Layer_Security]
4. Chhatra Thapa, Transport Layer Security (Slides)
   [https://www.slideshare.net/chhatrathapa1/avl-presentation-15664106]

Lịch sử phát triển

- SSL 1.0 (1994): Netscape, never publicly released
- SSL 2.0 (1995): many security flaws
- SSL 3.0 (1996): POODLE, weak RC4
- TLS 1.0 (1999): IETF, to be prohibited
  - incompatible with SSL 3.0
  - can be downgraded to SSL 3.0
- TLS 1.1 (2006): OK
- TLS 1.2 (2008): OK
- TLS 1.3 (10/2018): OK

Một số tấn công vào SSL/TLS

- **+ *BEAST*** *(CVE-2011-3389)* – ***2011****:* SSL 3.0 và TLS 1.0 (mã hóa CBC) => Nên sử dụng TLS 1.1, 1.2
- **+ *CRIME*** *(CVE-2012-4929)* – ***2012*** *:* Là một điểm yếu trong quá trình nén của TLS (TLS 1.0, 1.1, 1.2) => Nâng cấp trình duyệt lên phiên bản mới nhất
- **+ *BREACH*** *(CVE-2013-3587)* – ***2013*** *:* Vào TLS,

Gần giống CRIME tuy nhiên dựa trên điểm yếu là quá trình nén của HTTP => Nên bỏ phương thức nén trong HTTP, …

- ***POODLE*** *(CVE-2014-3566)* - ***2014*** *(Padding Oracle On Downgraded Legacy Encryption (POODLE)* => SSL 3.0
- **+ *Heartbleed*** *(CVE-2014-0160)* -***2014****:* OpenSSL, TLS => Ngăn chặn bằng cách cập nhật phiên bản mới nhất của OpenSSL
- + **SWEET32 (Birthday attacks against TLS ciphers with 64bit block size (CVE-2016-2183))**
- **+ROBOT (2017):** Là một lỗ hổng đã tồn tại 19 năm liên quan đến quá trình mã hóa và ký số RSA với khóa riêng trong các máy chủ hỗ trợ bảo mật với giao thức SSL/TLS.

Là điểm yếu cho phép thực hiện giải mã RSA và thực hiện các hoạt động ký với khóa bí mật của một máy chủ TLS (Tấn công chọn bản mã RSA trong định dạng PKCS#1)

….

Các dịch vụ của SSL/TLS

- Xác thực thực thể (1 chiều hoặc 2 chiều)
- Thỏa thuận bộ tham số mật mã
- Trao đổi khóa phiên
- Nén dữ liệu
- Bí mật dữ liệu
- Xác thực (và toàn vẹn) dữ liệu

Giao thức trao đổi khóa

Algorithm	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3
RSA	Yes	Yes	Yes	Yes	No
DH-RSA	Yes	Yes	Yes	Yes	No
DHE-RSA (*)	Yes	Yes	Yes	Yes	Yes
ECDH-RSA	No	Yes	Yes	Yes	No
ECDHE-RSA (*)	No	Yes	Yes	Yes	Yes
DH-DSS	Yes	Yes	Yes	Yes	No
DHE-DSS (*)	Yes	Yes	Yes	Yes	No
ECDH-ECDSA	No	Yes	Yes	Yes	No
ECDHE-ECDSA (*)	No	Yes	Yes	Yes	Yes
PSK	No	Yes	Yes	Yes	
PSK-RSA	No	Yes	Yes	Yes	
DHE-PSK (*)	No	Yes	Yes	Yes	
ECDHE-PSK (*)	No	Yes	Yes	Yes	
Kerberos	No	Yes	Yes	Yes	

Thuật toán mã hóa

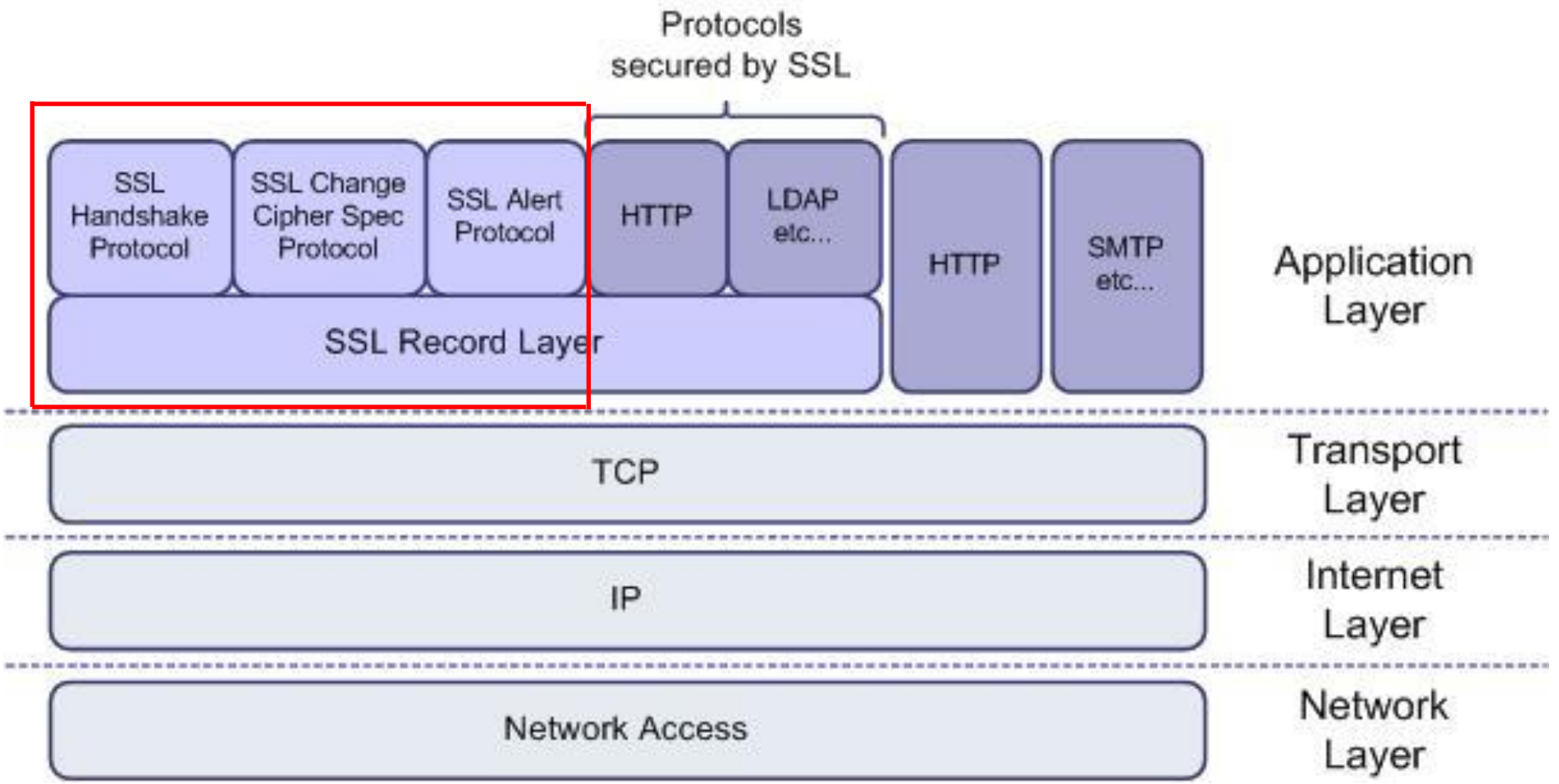
Algorithm	Key length	SSL 3.0	TLS 1.1	TLS 1.2	TLS 1.3
AES GCM	256, 128	N/A	N/A	Secure	Secure
AES CCM		N/A	N/A	Secure	Secure
AES CBC		N/A	Secure	Secure	N/A
Camellia GCM	256, 128	N/A	N/A	Secure	Secure
Camellia CBC		N/A	Secure	Secure	N/A
ARIA GCM	256, 128	N/A	N/A	Secure	Secure
ARIA CBC		N/A	Secure	Secure	N/A
SEED CBC	128	N/A	Secure	Secure	N/A
...					

Toàn vẹn và Xác thực dữ liệu

Algorithm	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3
HMAC-MD5	Yes	Yes	Yes	Yes	No
HMAC-SHA1	Yes	Yes	Yes	Yes	No
HMAC-SHA256/384	No	No	No	Yes	No
AEAD	No	No	No	Yes	Yes
GOST 28147-89 IMIT	No	Yes	Yes	Yes	
GOST R 34.11-94	No	Yes	Yes	Yes	

AEAD = Authenticated Encryption with Associated Data

SSL in TCP/IP Stack



Kiến trúc không thay đổi qua các phiên bản

Các giao thức con của SSL

- **Giao thức SSL :** gồm 4 giao thức con:
  - SSL Hanshake Protocol: thực hiện chức năng thỏa thuận các thuật toán, tham số, trao đổi khóa, xác thực server và client (nếu có lựa chọn)
  - SSL Record Protocol: phân mảnh, nén, tính MAC, mã hóa dữ liệu
  - SSL Alert Protocol: thông báo lỗi
  - SSL Change Cipher Spec Protocol: thông báo xác nhận kết thúc giai đoạn HandShake Protocol

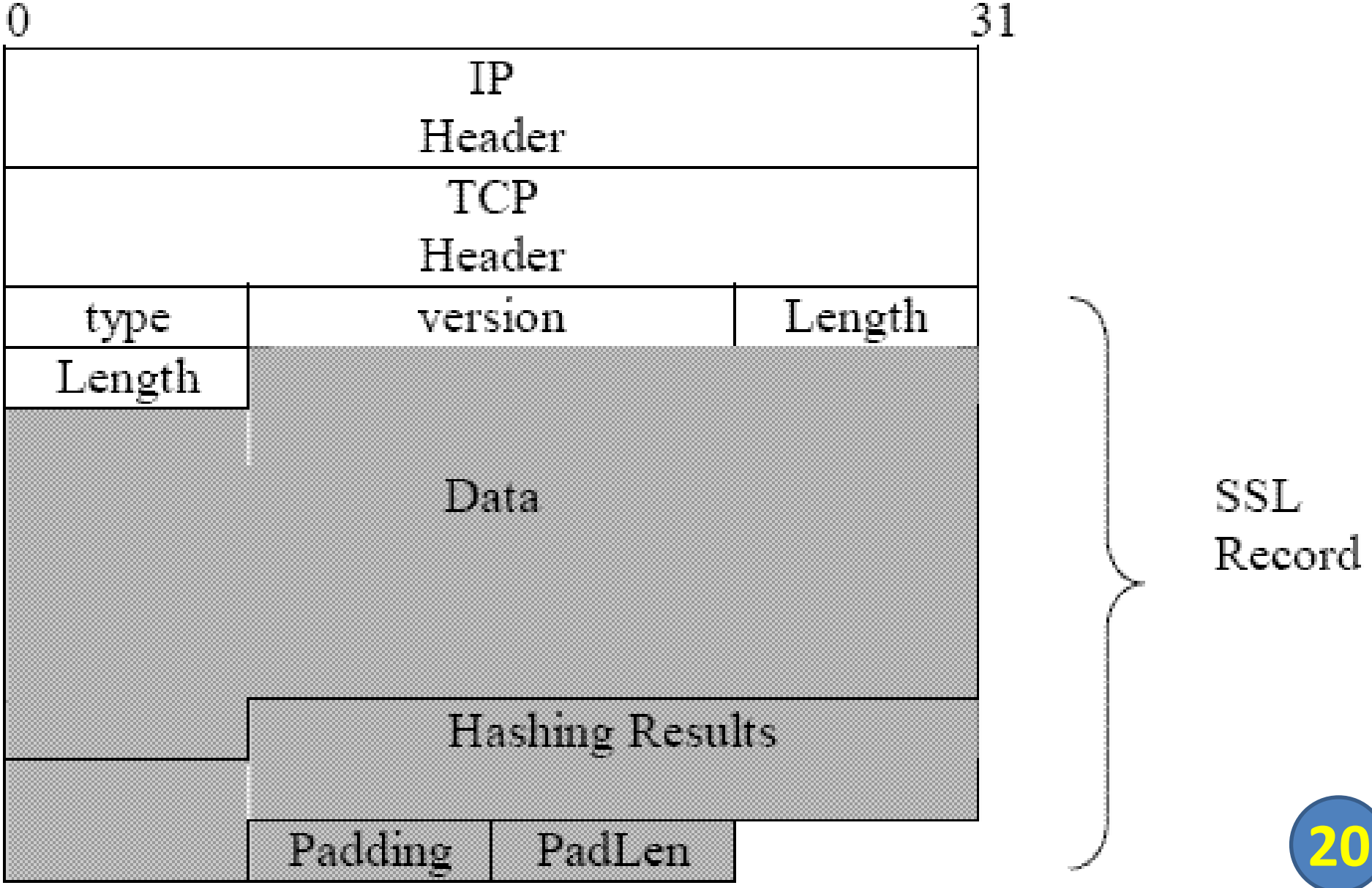
1 Tổng quan về SSL/TLS

2 Giao thức Record

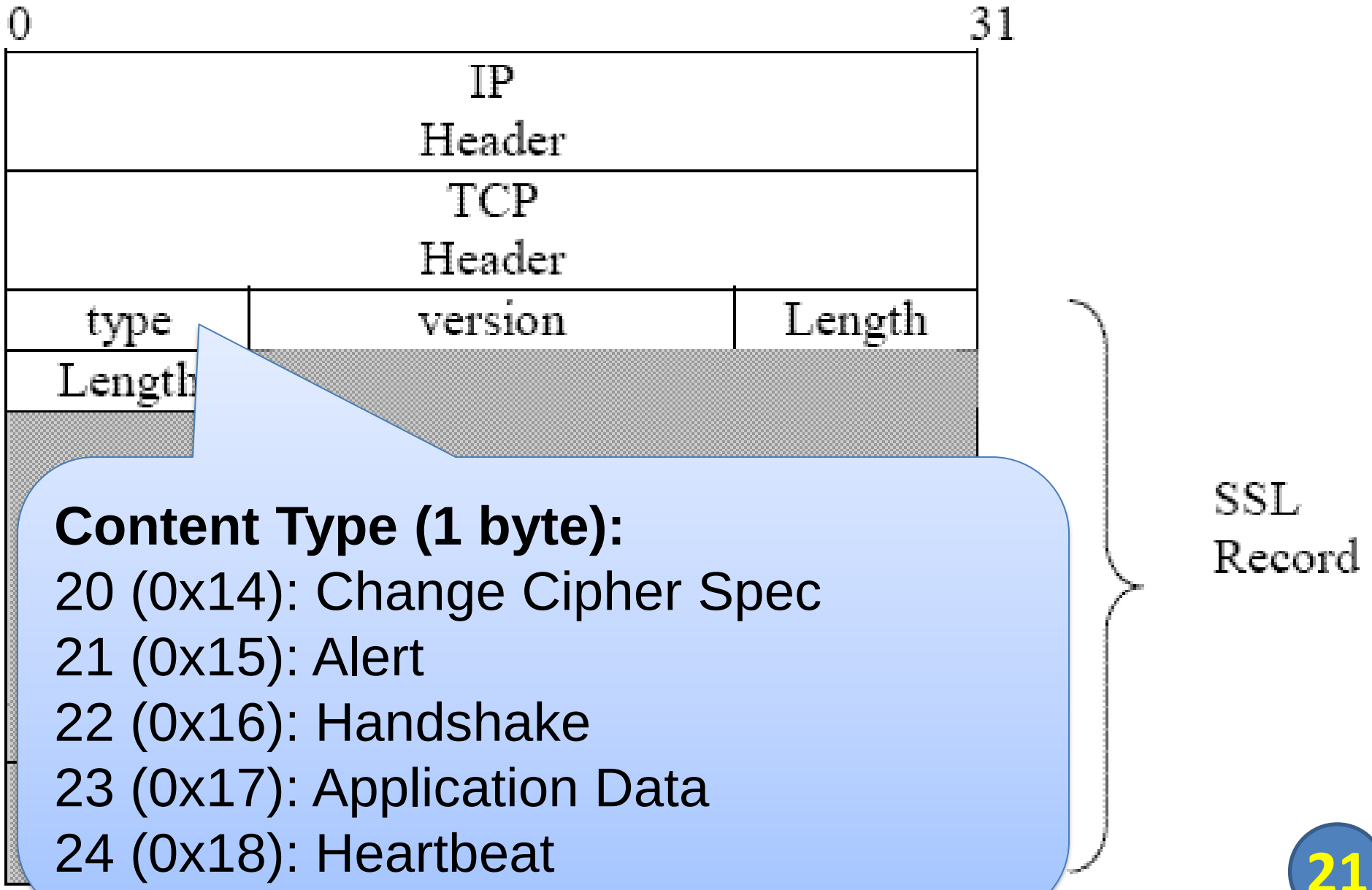
3 Giao thức Handshake

Giao thức Record

Định dạng gói tin SSL có chứa phần Record header



Giao thức Record



Content Type (1 byte):

20 (0x14): Change Cipher Spec

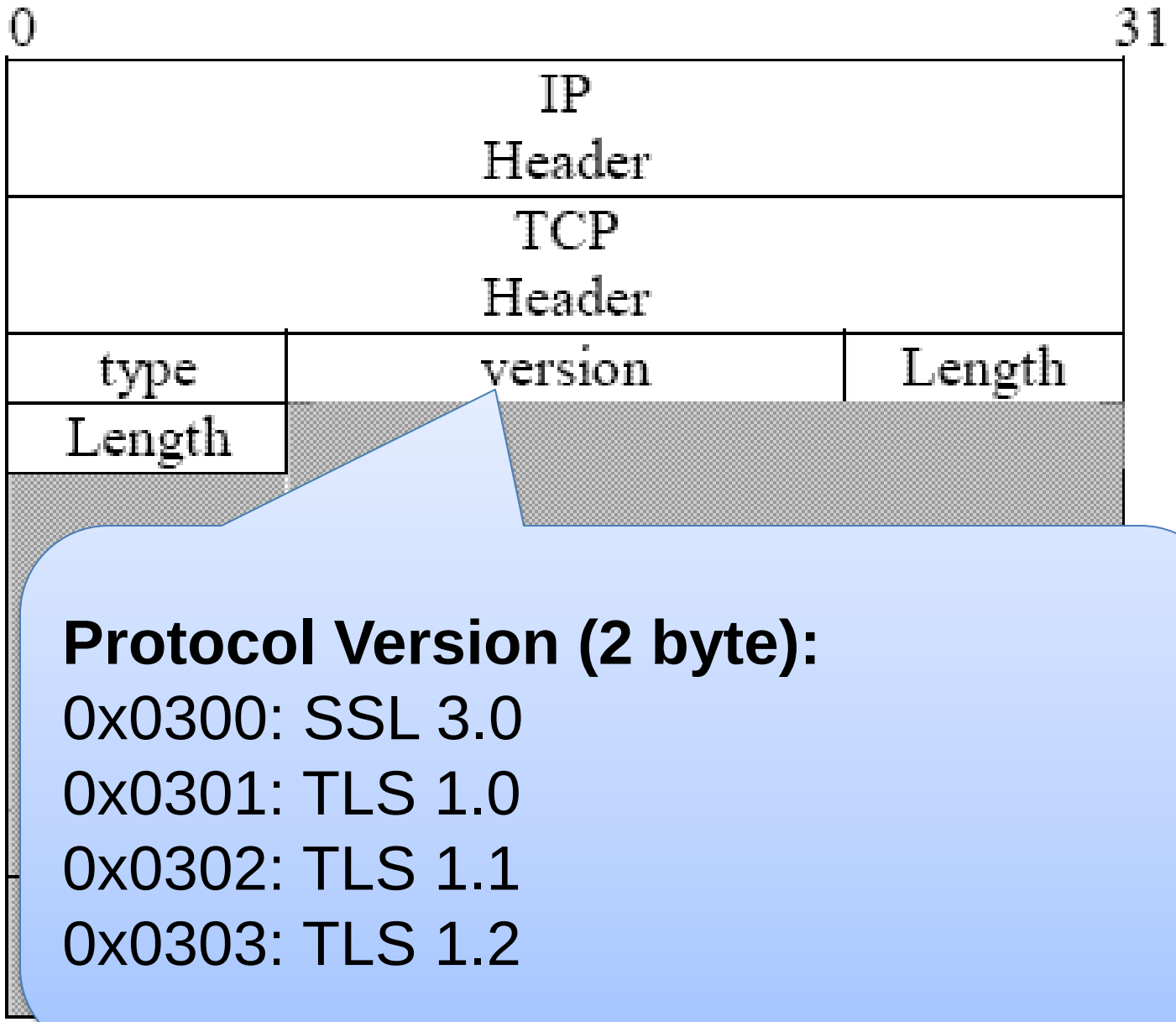
21 (0x15): Alert

22 (0x16): Handshake

23 (0x17): Application Data

24 (0x18): Heartbeat

Giao thức Record



Protocol Version (2 byte):

0x0300: SSL 3.0

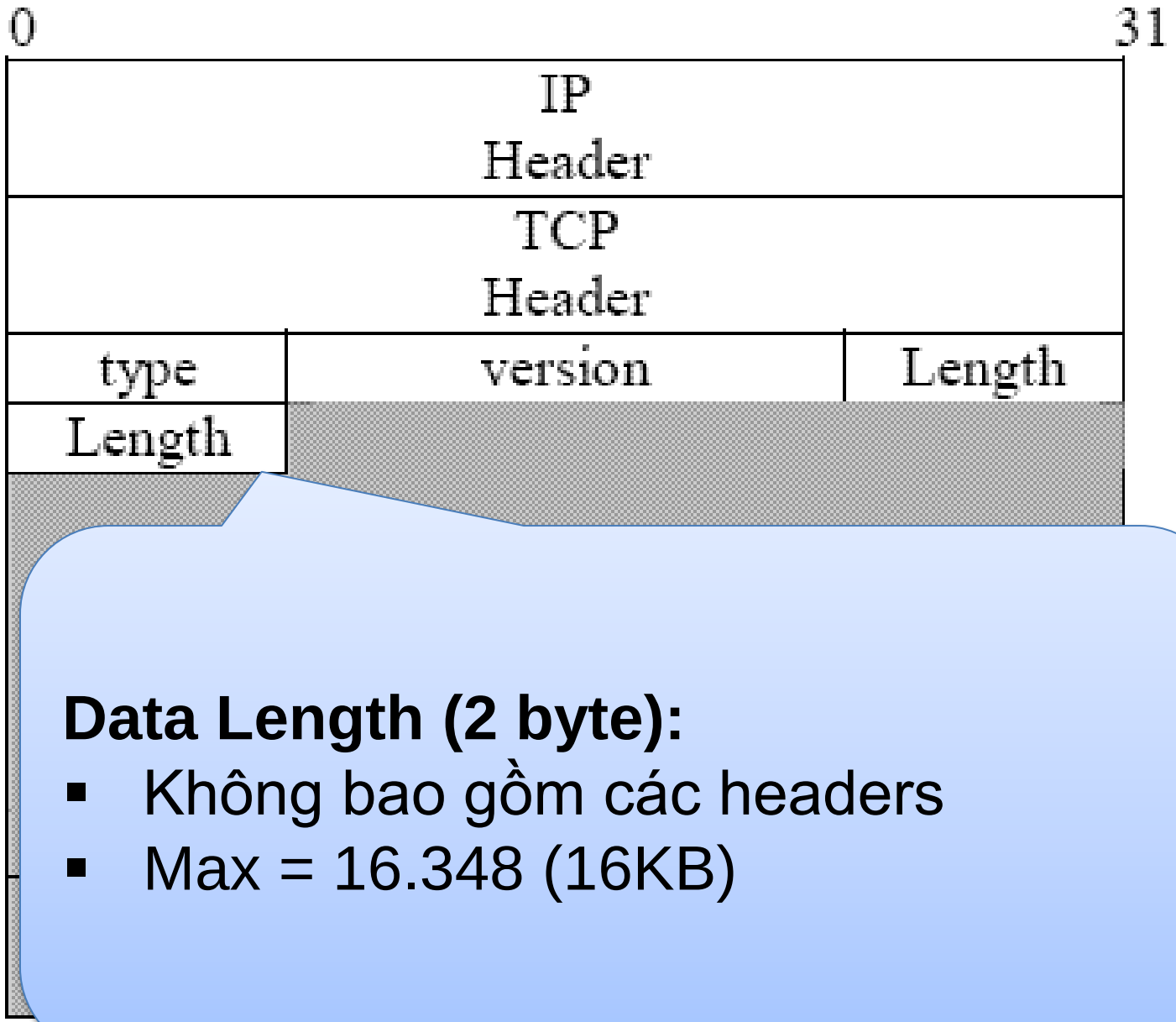
0x0301: TLS 1.0

0x0302: TLS 1.1

0x0303: TLS 1.2

SSL
Record

Giao thức Record



SSL
Record

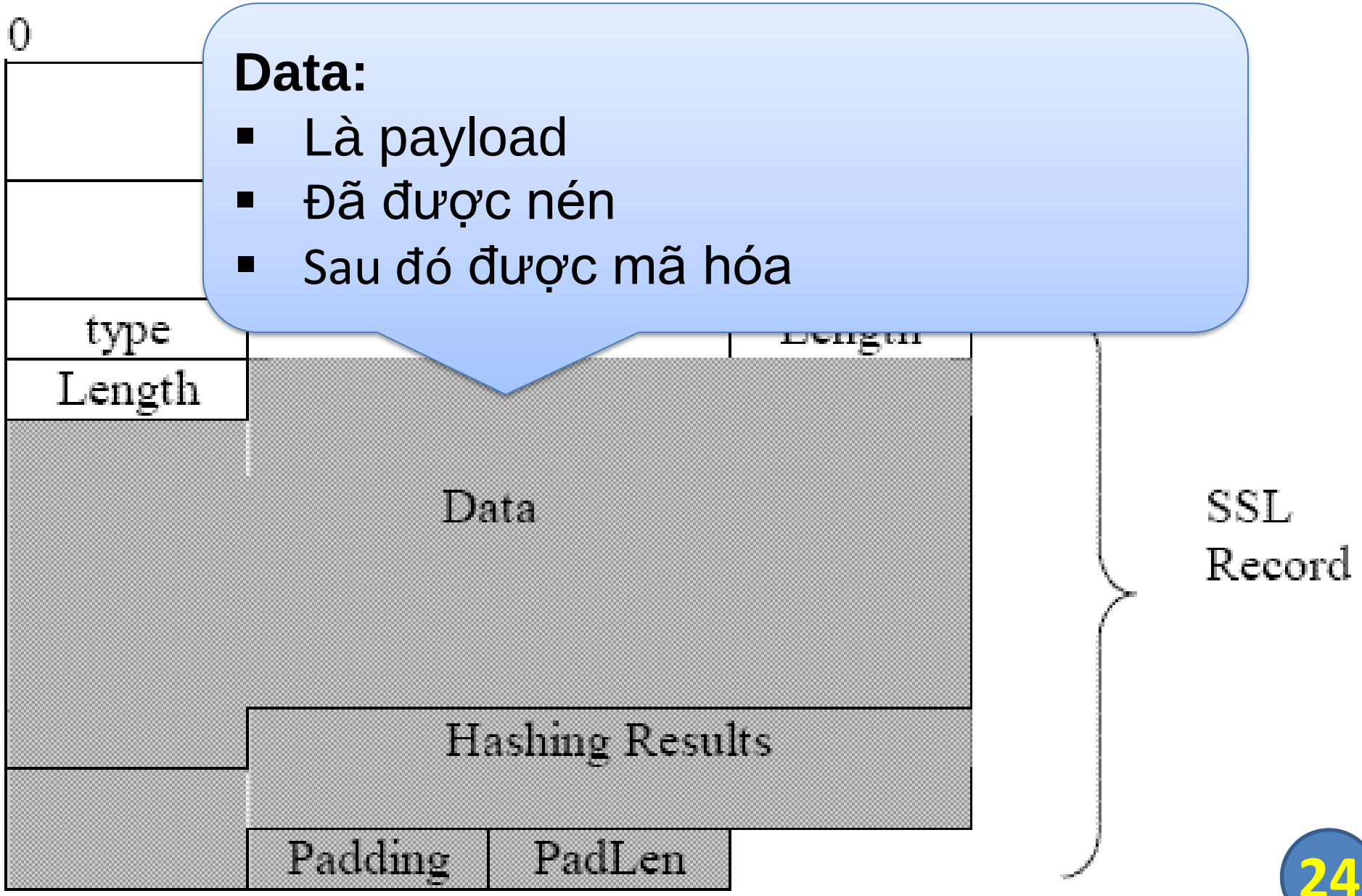
Data Length (2 byte):

- Không bao gồm các headers
- Max = 16.348 (16KB)

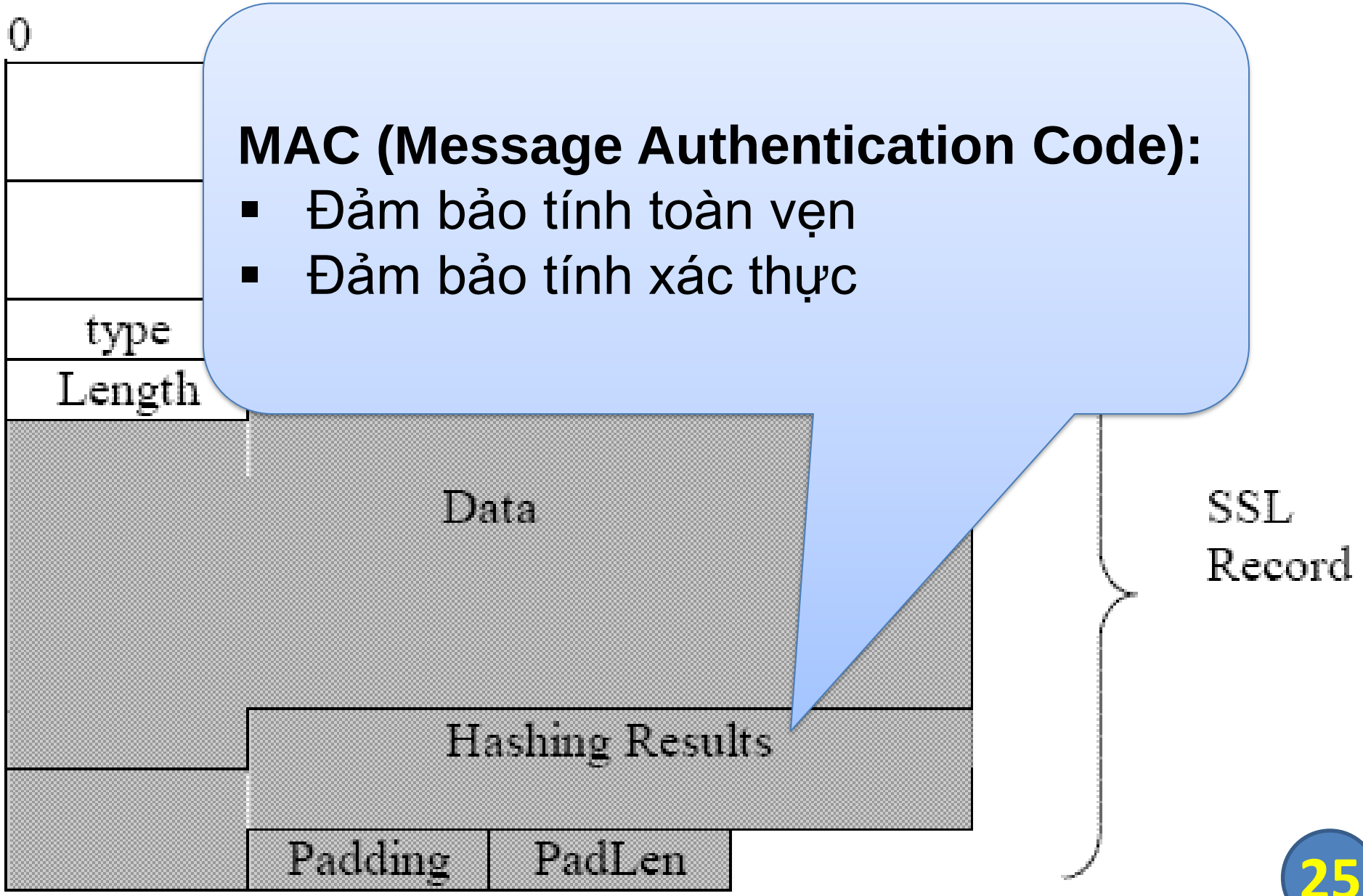
Giao thức Record

Data:

- Là payload
- Đã được nén
- Sau đó được mã hóa



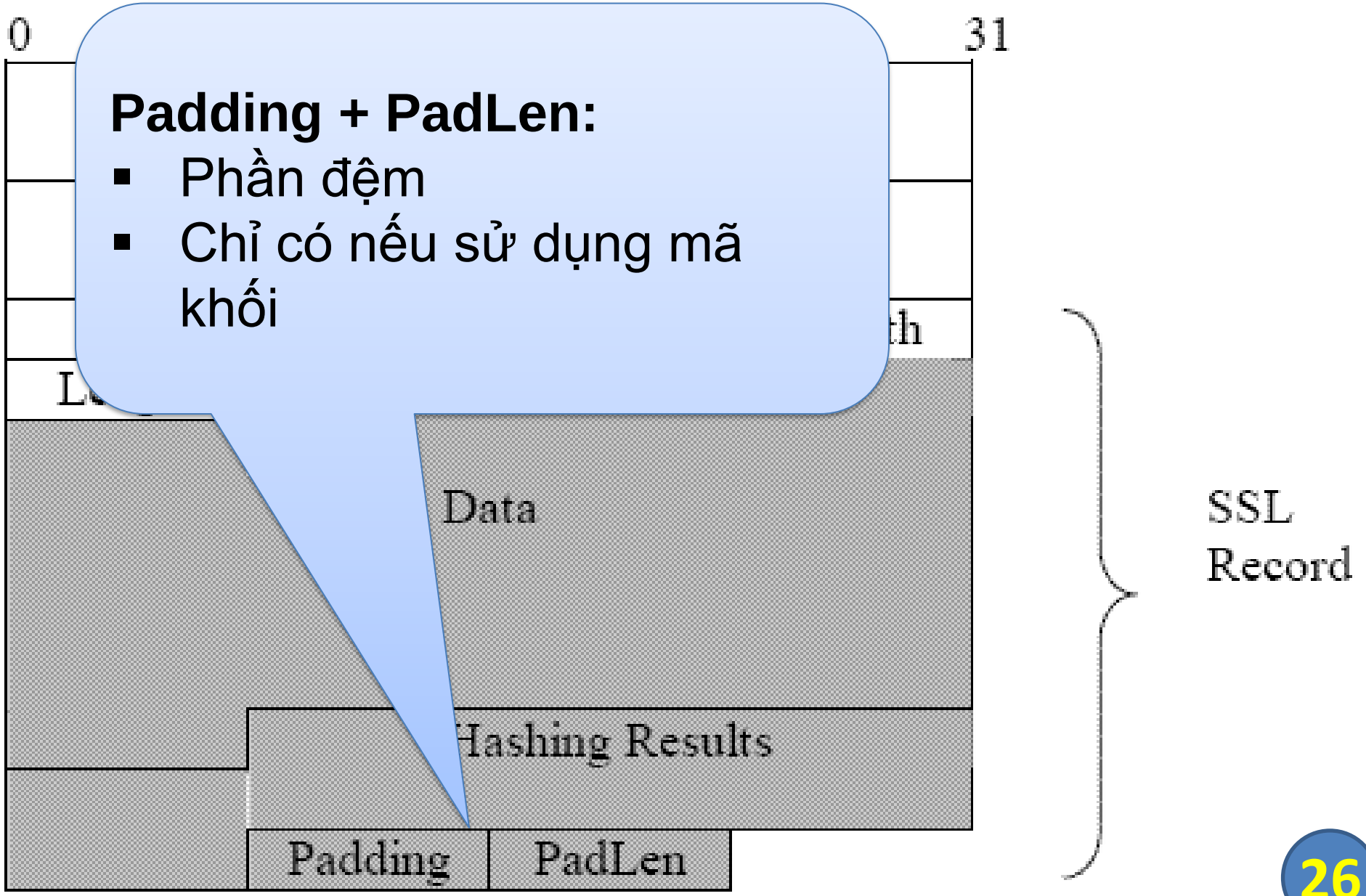
Giao thức Record



MAC (Message Authentication Code):

- Đảm bảo tính toàn vẹn
- Đảm bảo tính xác thực

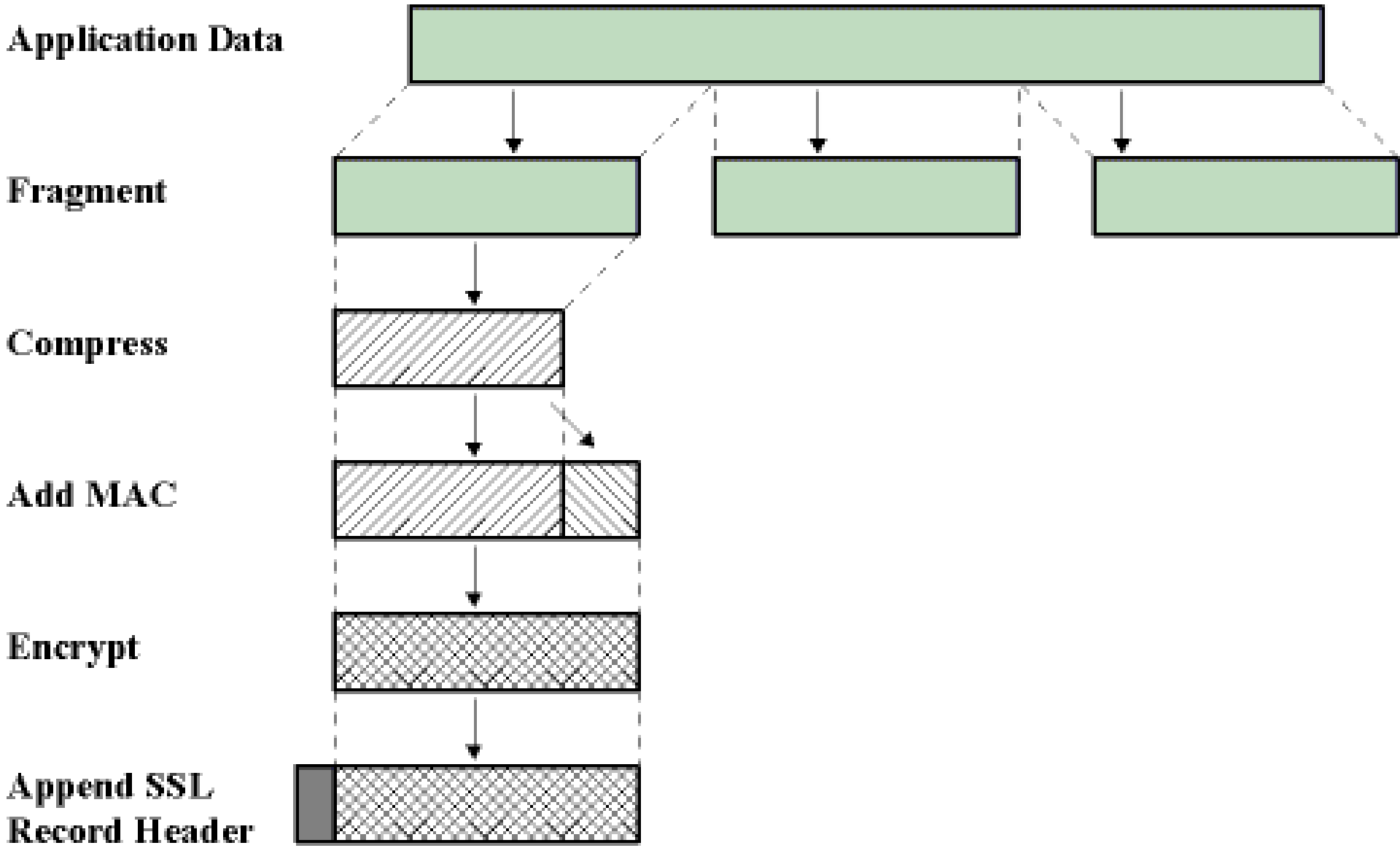
Giao thức Record



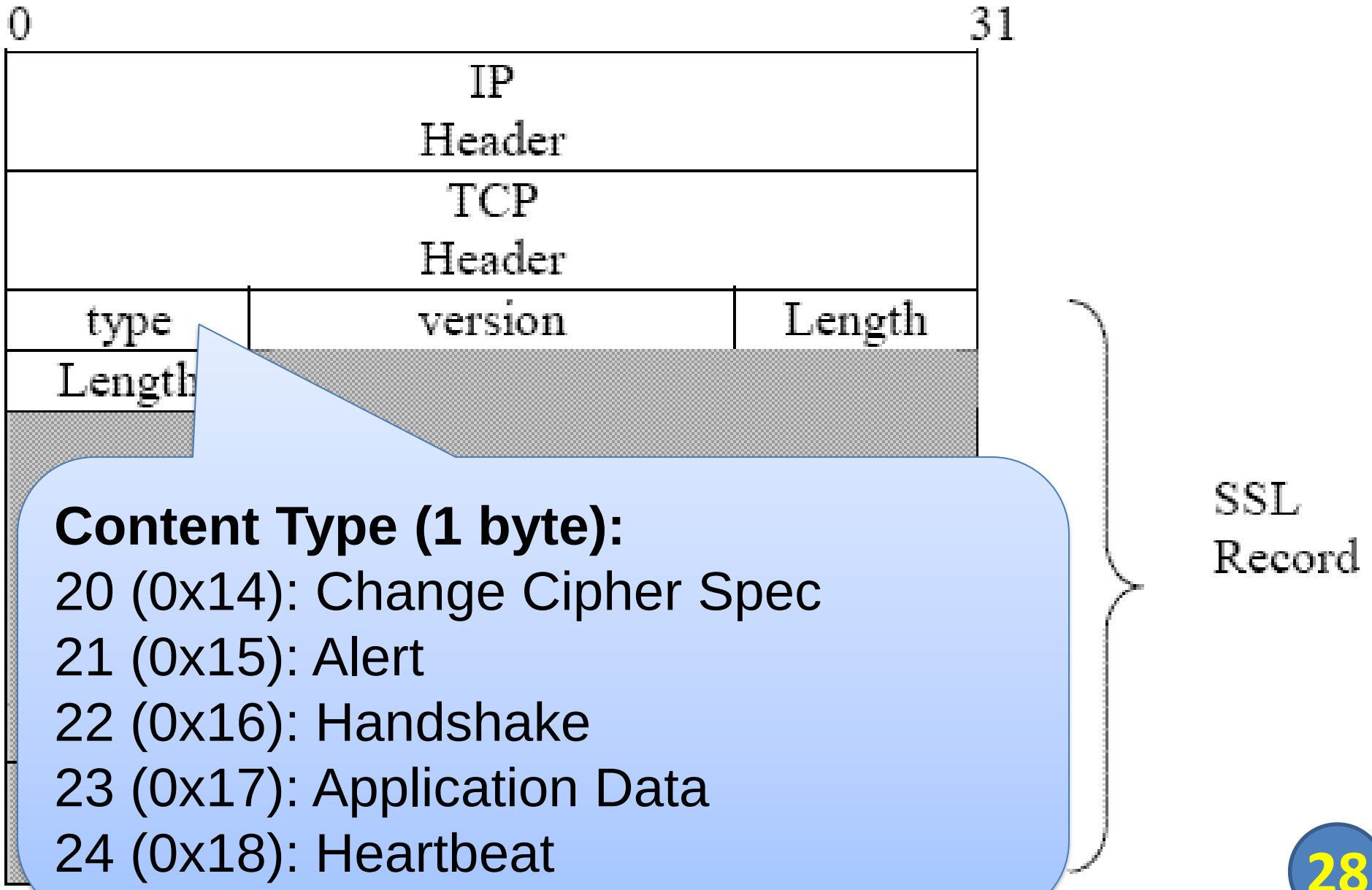
Padding + PadLen:

- Phần đệm
- Chỉ có nếu sử dụng mã khối

Record Protocol Operation



Giao thức Record



Content Type (1 byte):

20 (0x14): Change Cipher Spec

21 (0x15): Alert

22 (0x16): Handshake

23 (0x17): Application Data

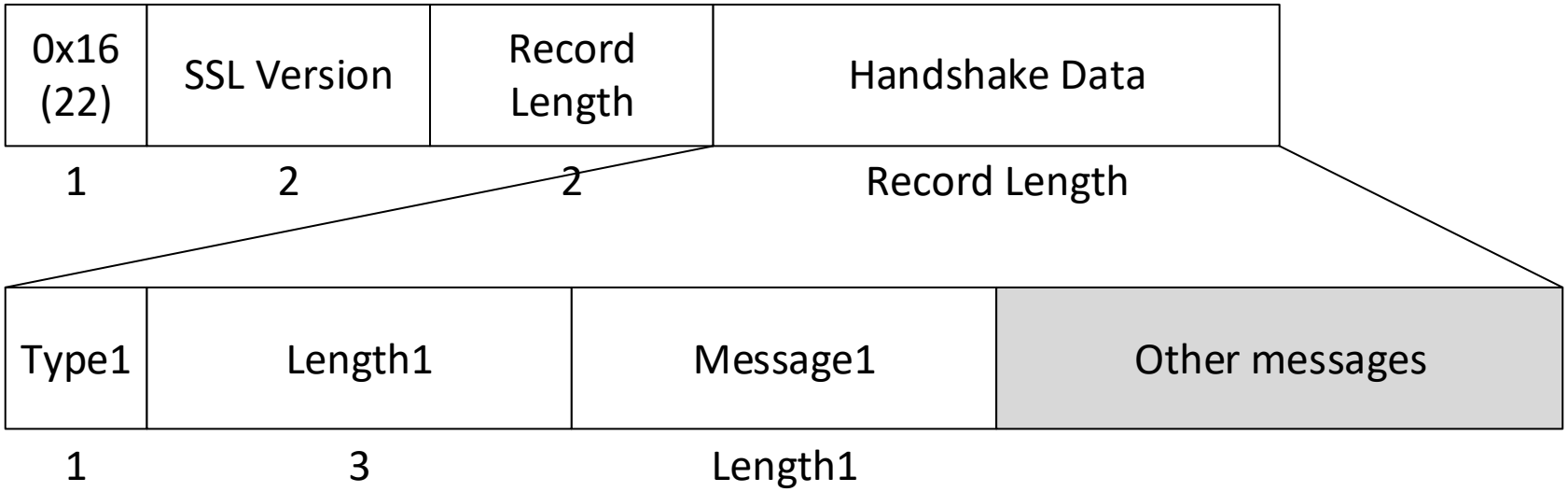
24 (0x18): Heartbeat

1 Tổng quan về SSL/TLS

2 Giao thức Record

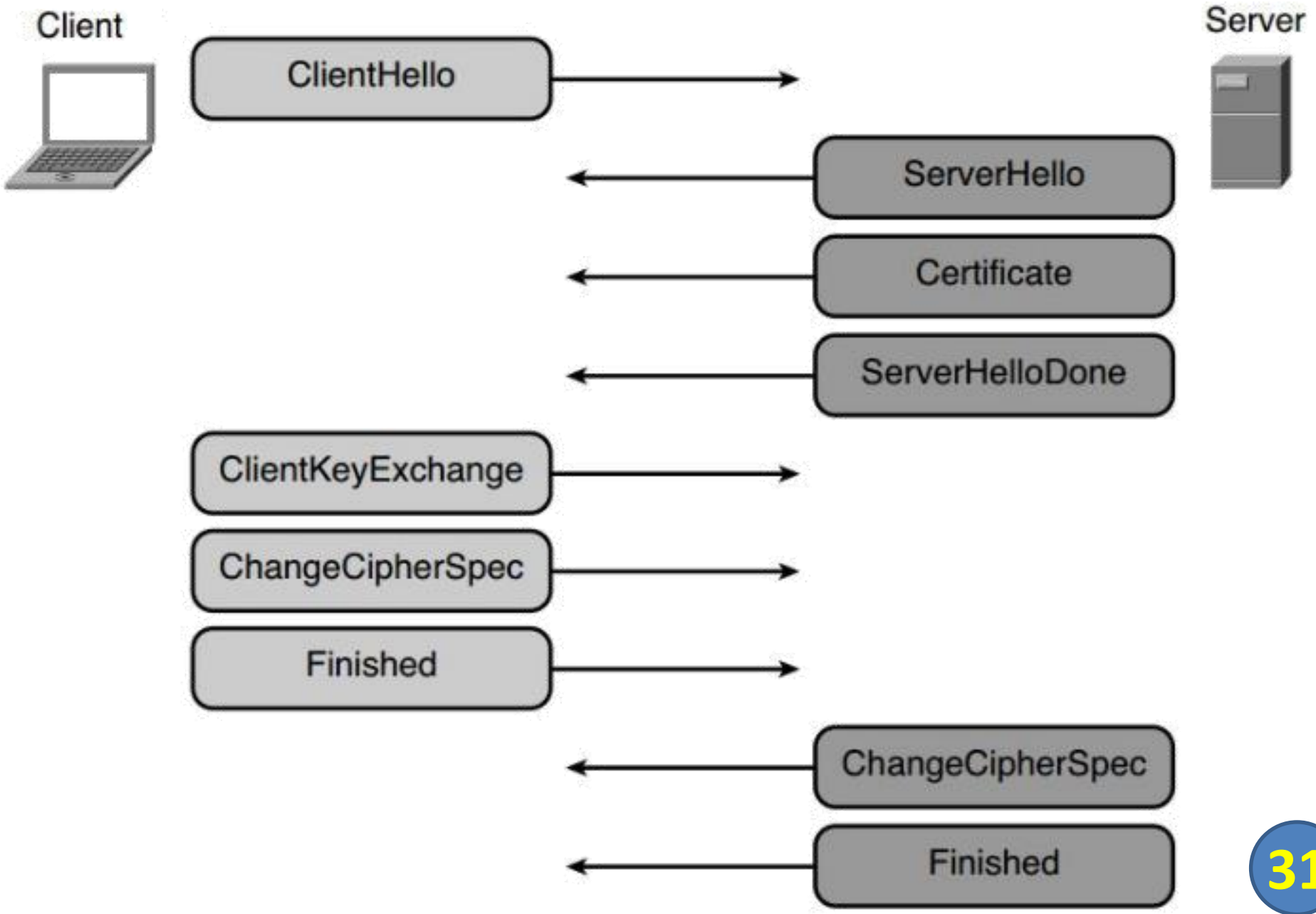
3 Giao thức Handshake

Giao thức Handshake (0x16)

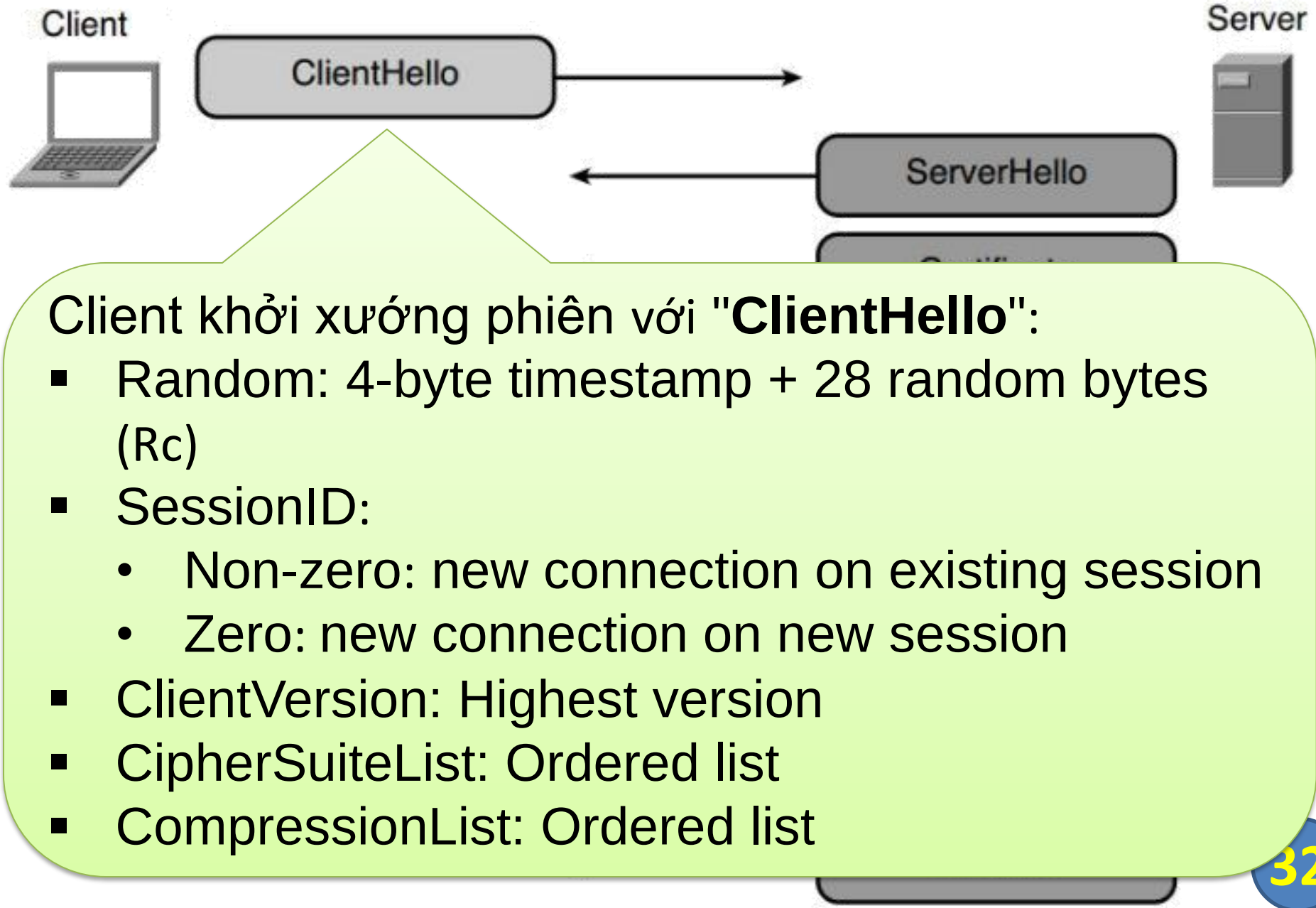


Handshake Type	dec	hex
HELLO_REQUEST	0	0x00
CLIENT_HELLO	1	0x01
SERVER_HELLO	2	0x02
CERTIFICATE	11	0x0b
SERVER_KEY_EXCHANGE	12	0x0c
CERTIFICATE_REQUEST	13	0x0d
SERVER_DONE	14	0x0e
CERTIFICATE_VERIFY	15	0x0f
CLIENT_KEY_EXCHANGE	16	0x10
FINISHED	20	0x14

Giao thức Handshake



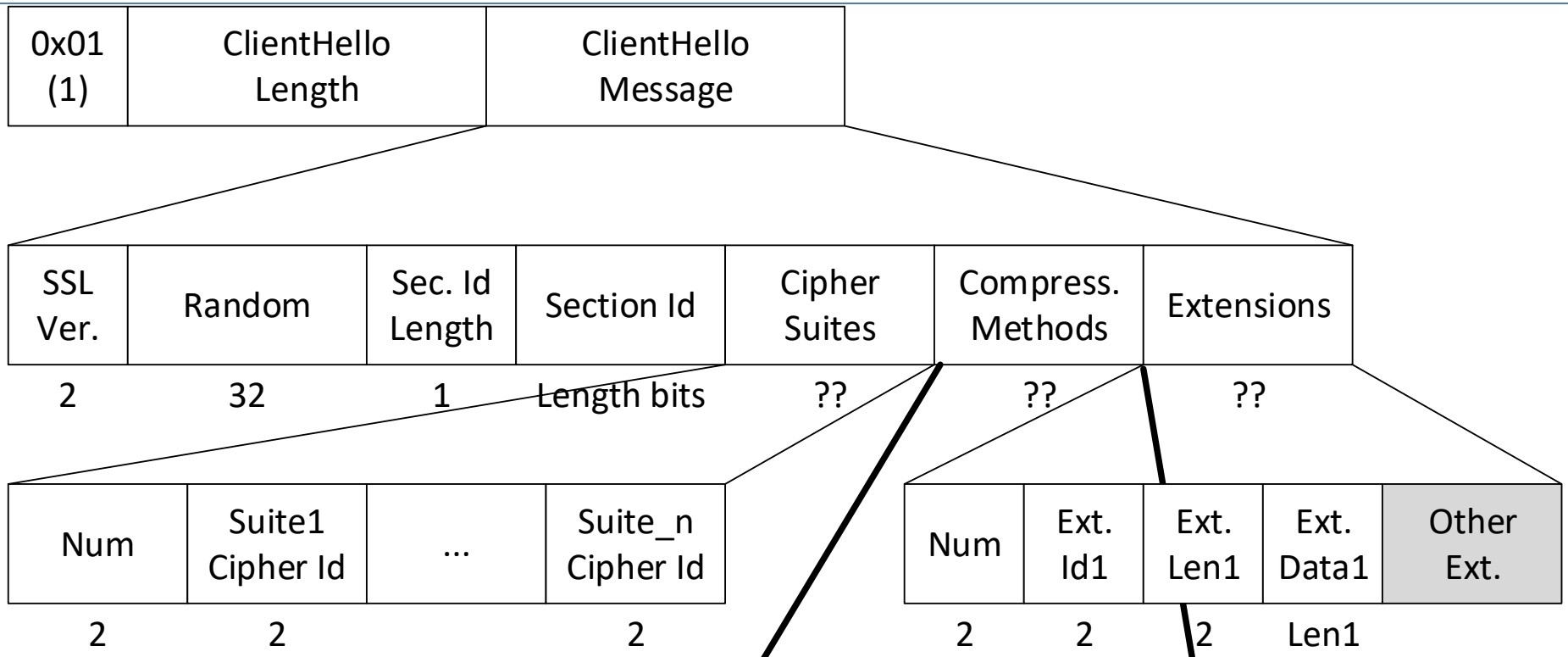
Giao thức Handshake



Client khởi xướng phiên với "**ClientHello**":

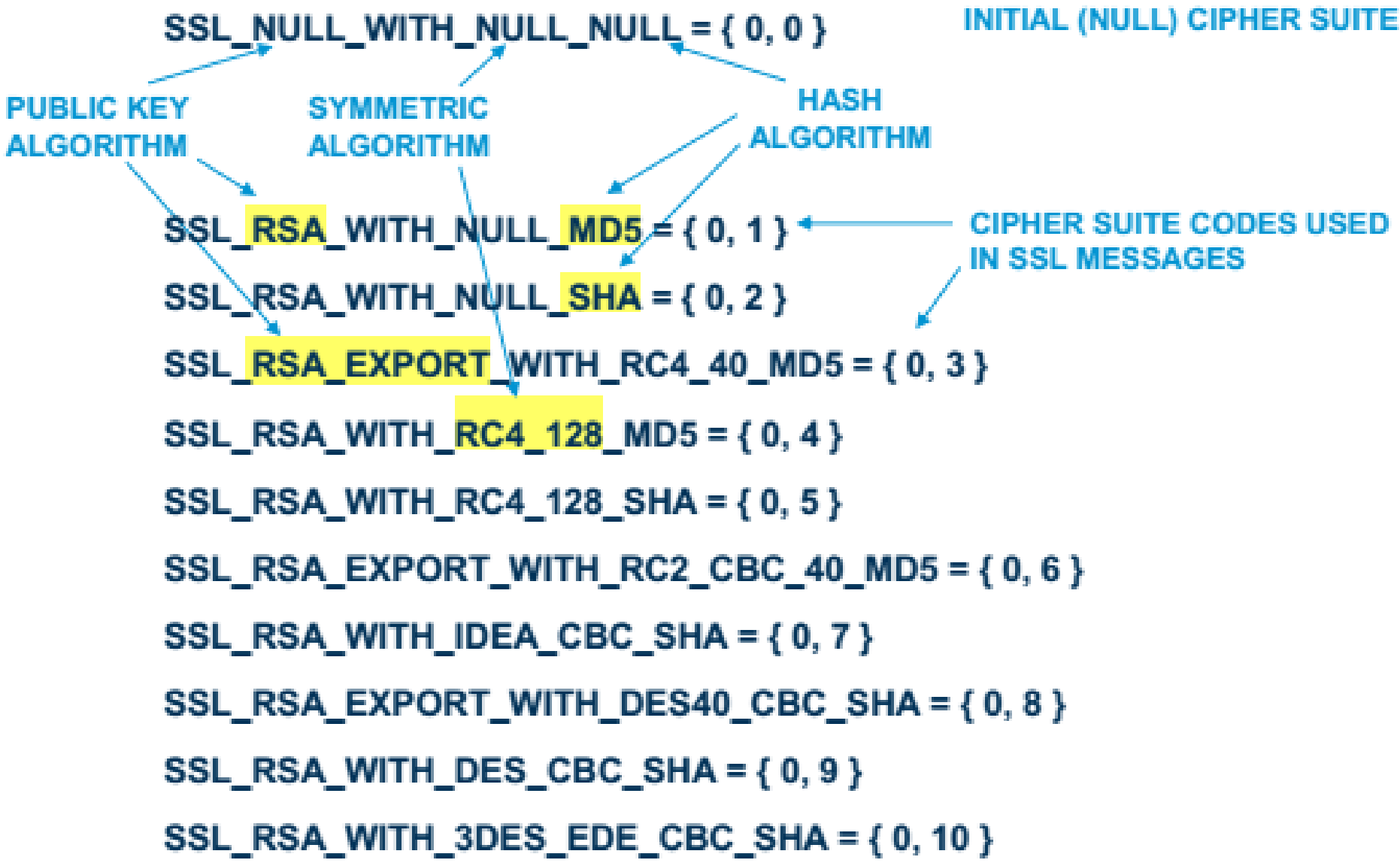
- Random: 4-byte timestamp + 28 random bytes (Rc)
- SessionID:
  - Non-zero: new connection on existing session
  - Zero: new connection on new session
- ClientVersion: Highest version
- CipherSuiteList: Ordered list
- CompressionList: Ordered list

ClientHello Message (0x01)




```
struct {
        ProtocolVersion client_version;
        Random random;
        SessionID session_id;
        CipherSuite cipher_suites<2..2^16-1>;
        CompressionMethod compression_methods<1..2^8-1>;
        Extension extensions<0..2^16-1>;
} ClientHello;
```

ClientHello Message: Cipher Suites

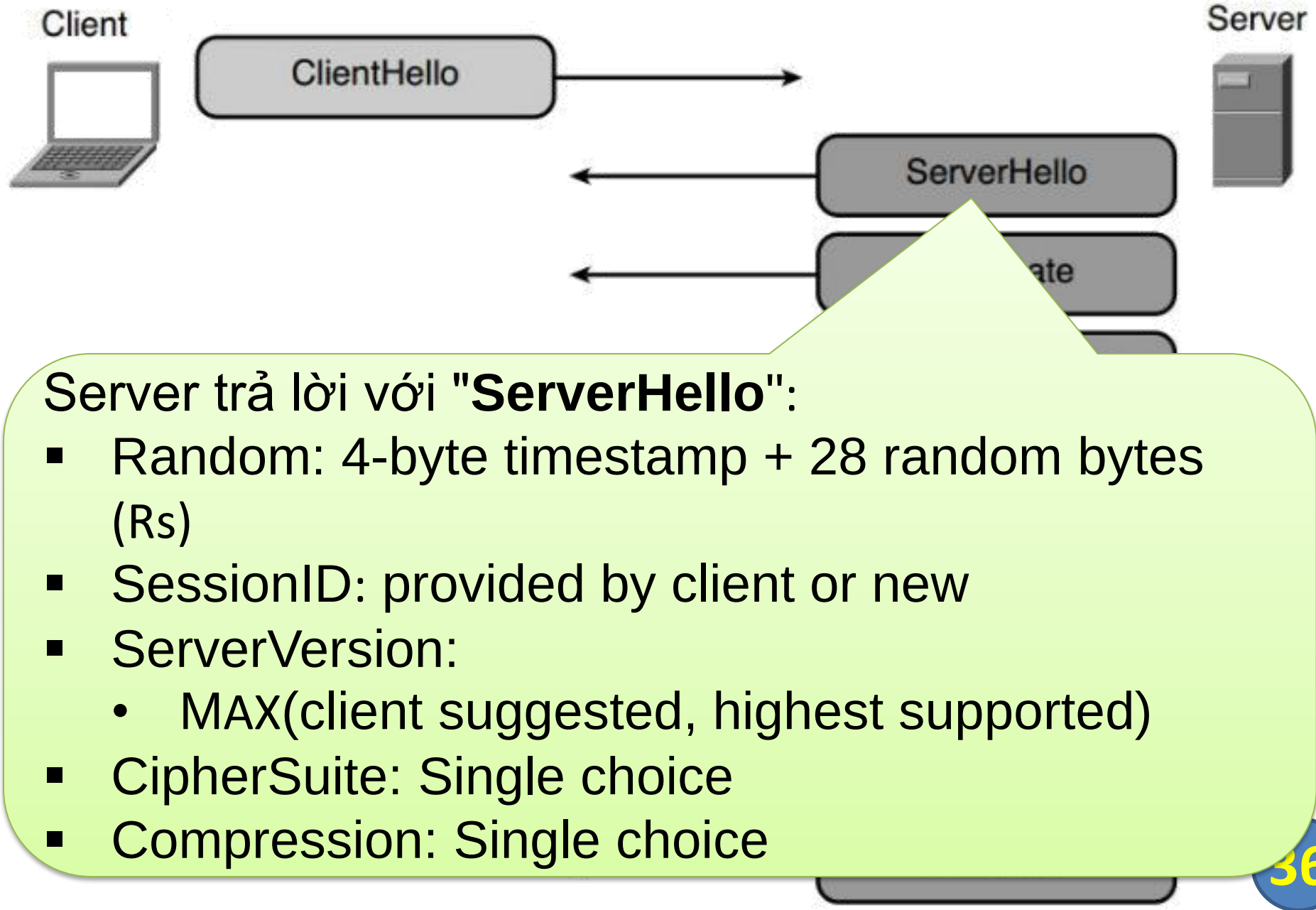


ClientHello Message: TLS 1.3 Cipher Suites

CipherSuite TLS_AEAD_HASH = VALUE;

TLS_AES_128_GCM_SHA256	0x13,0x01}
TLS_AES_256_GCM_SHA384	{0x13,0x02}
TLS_CHACHA20_POLY1305_SHA256	{0x13,0x03}
TLS_AES_128_CCM_SHA256	{0x13,0x04}
TLS_AES_128_CCM_8_SHA256	{0x13,0x05}

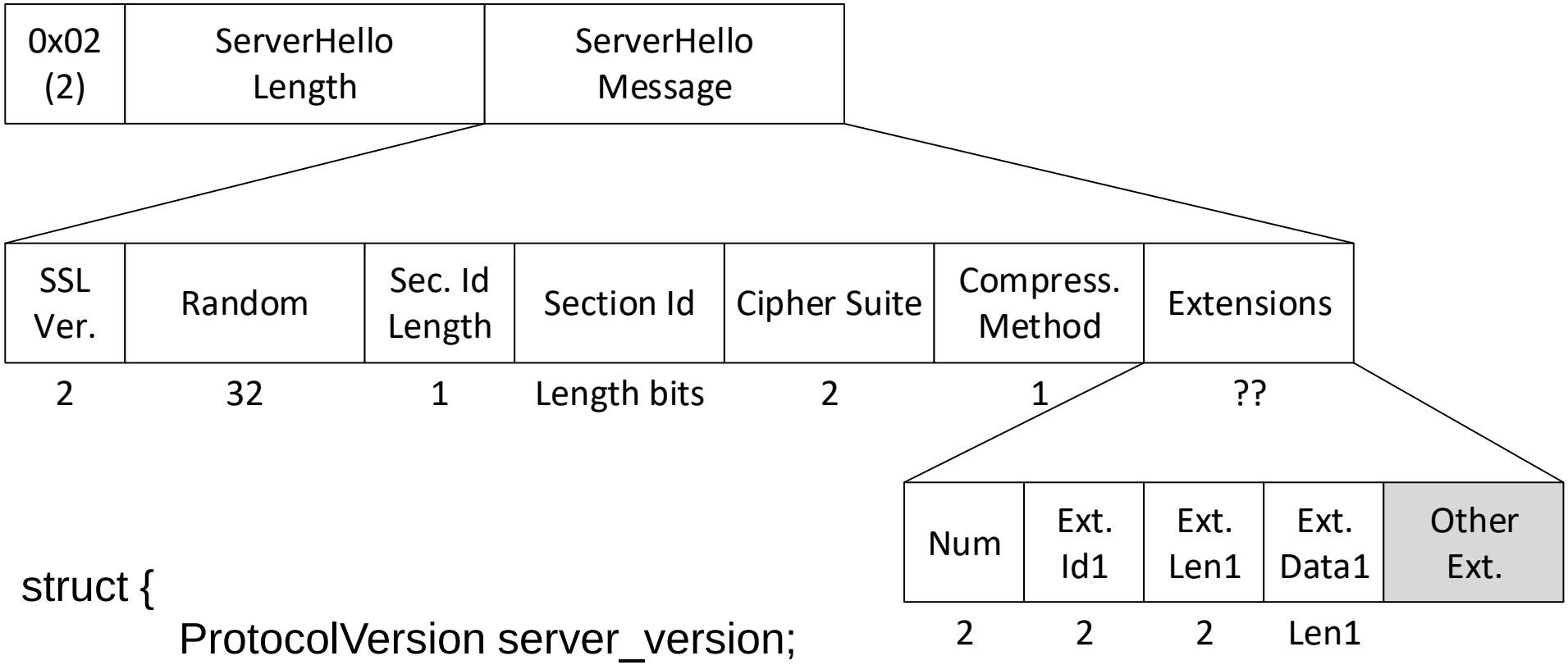
Giao thức Handshake



Server trả lời với "**ServerHello**":

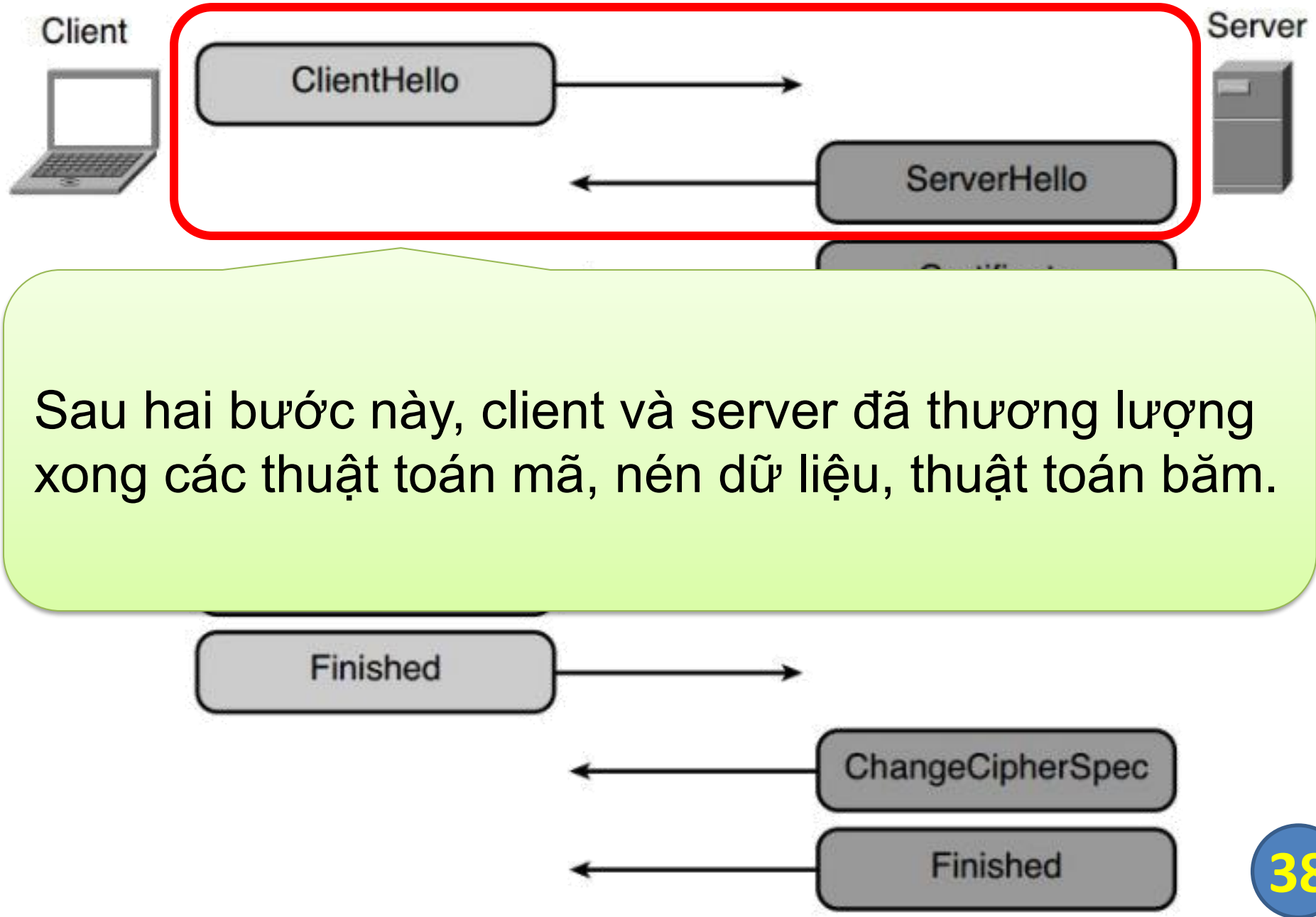
- Random: 4-byte timestamp + 28 random bytes (Rs)
- SessionID: provided by client or new
- ServerVersion:
  - MAX(client suggested, highest supported)
- CipherSuite: Single choice
- Compression: Single choice

ServerHello Message (0x02)



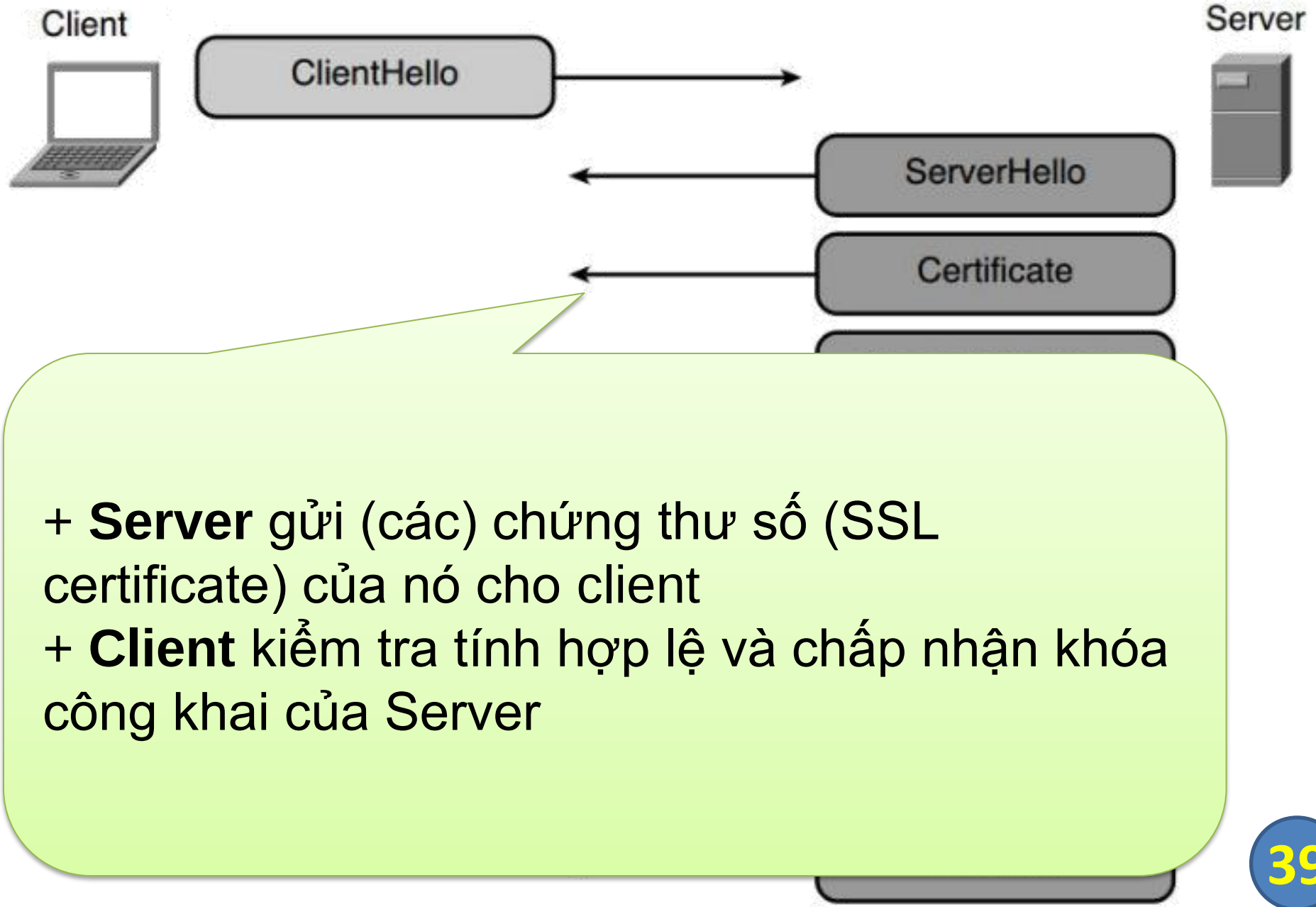

```
struct {
        ProtocolVersion server_version;
        Random random;
        SessionID session_id;
        CipherSuite cipher_suite;
        CompressionMethod compression_method;
        Extension extensions<0..2^16-1>;
} ServerHello;
```

Giao thức Handshake



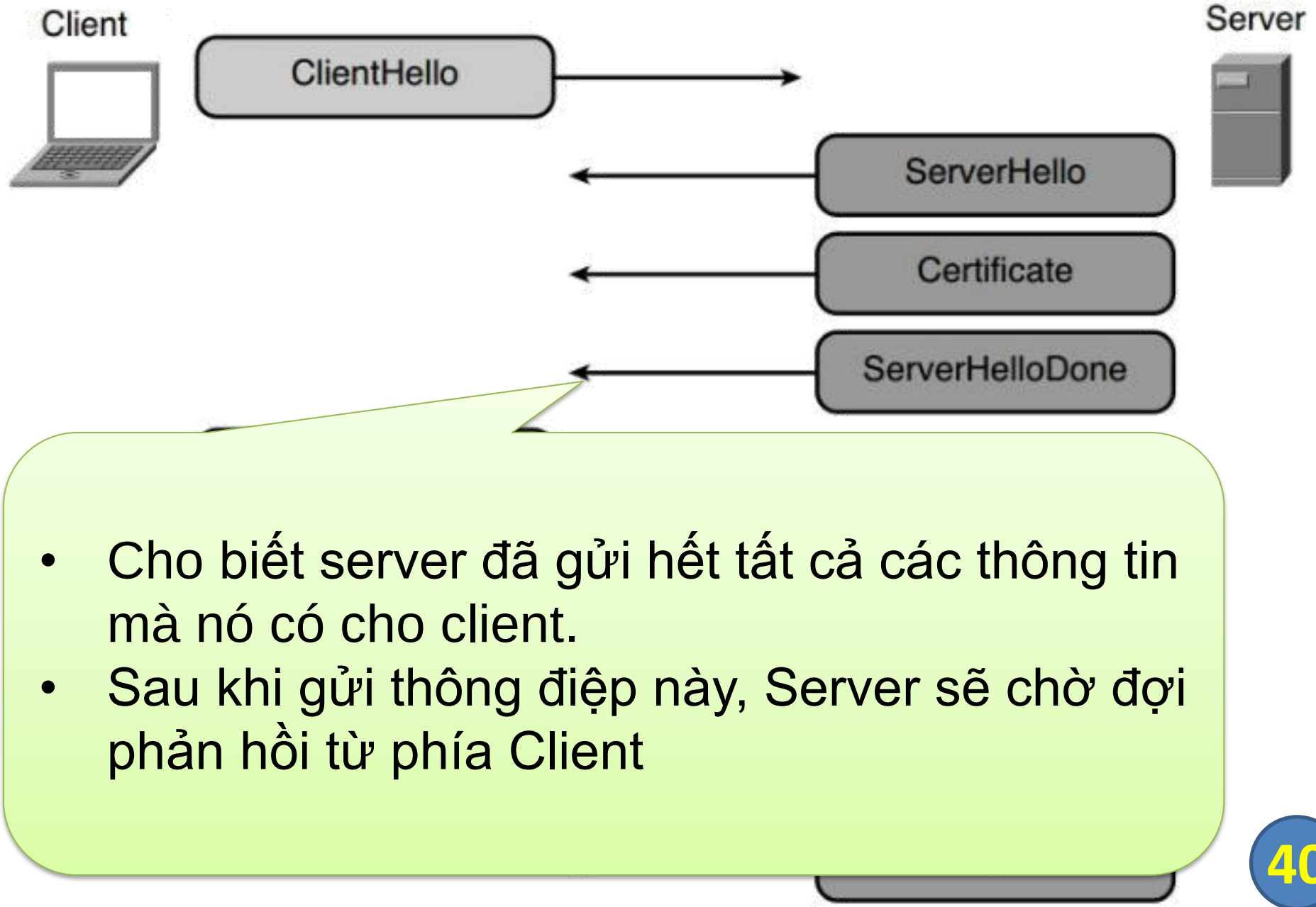
Sau hai bước này, client và server đã thương lượng xong các thuật toán mã, nén dữ liệu, thuật toán băm.

Giao thức Handshake



+ **Server** gửi (các) chứng thư số (SSL certificate) của nó cho client
+ **Client** kiểm tra tính hợp lệ và chấp nhận khóa công khai của Server

Giao thức Handshake



- Cho biết server đã gửi hết tất cả các thông tin mà nó có cho client.
- Sau khi gửi thông điệp này, Server sẽ chờ đợi phản hồi từ phía Client

Giao thức Handshake

Client:

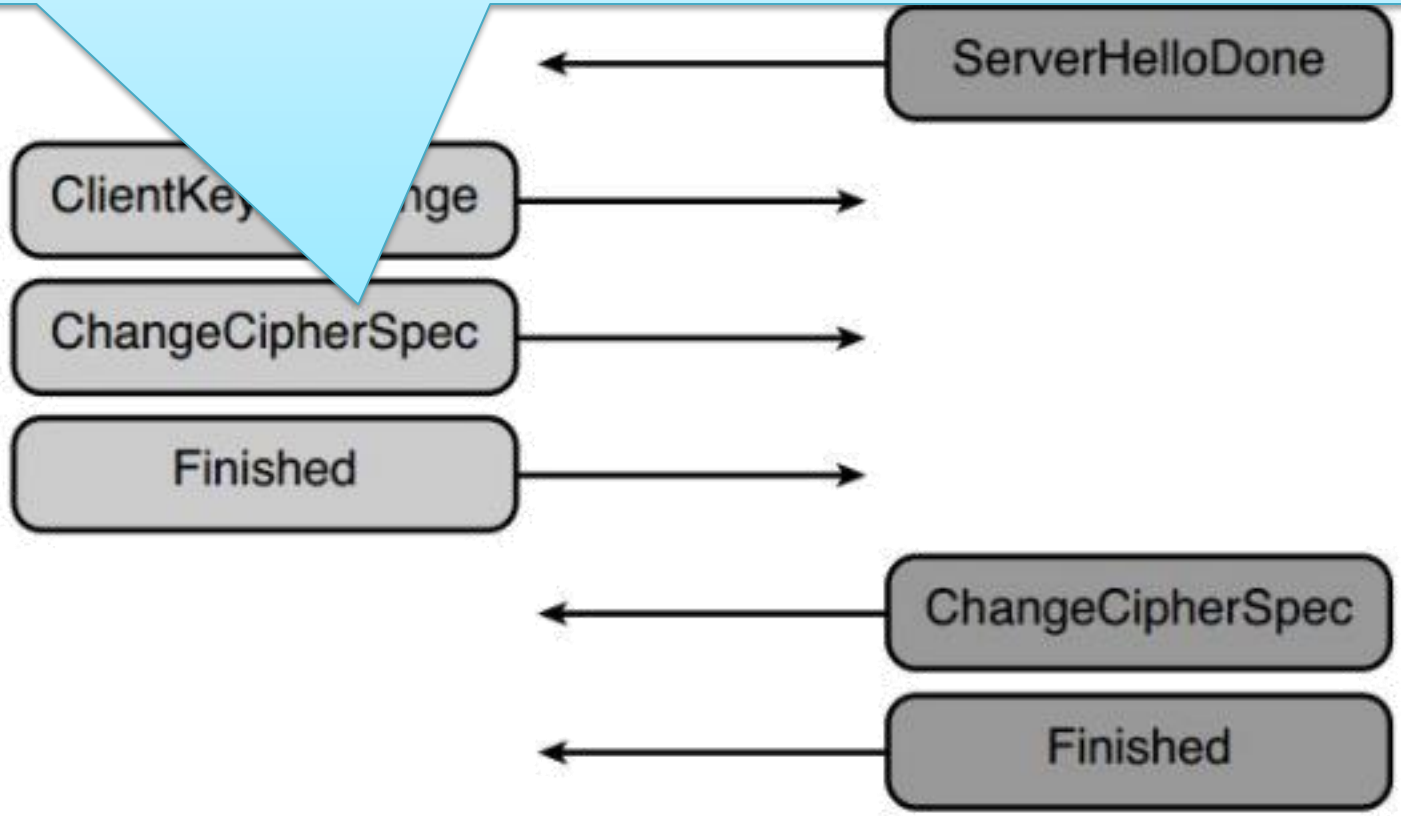
- Sinh một Pre_Master_Secret (PMK) (48 bytes)
- Mã hóa nó bằng public key của server ➔ Về sau, nếu Server giải mã được thì Server coi như được xác thực
- Thuật toán mã hóa đã thương lượng (VD: RSA)

Server:

- Giải mã bằng private key của mình.
- Bây giờ cả client và server đã có: PMK, Rc, Rs
- (PMK, Rc, Rs) ➔ Master Key ➔ Session Keys

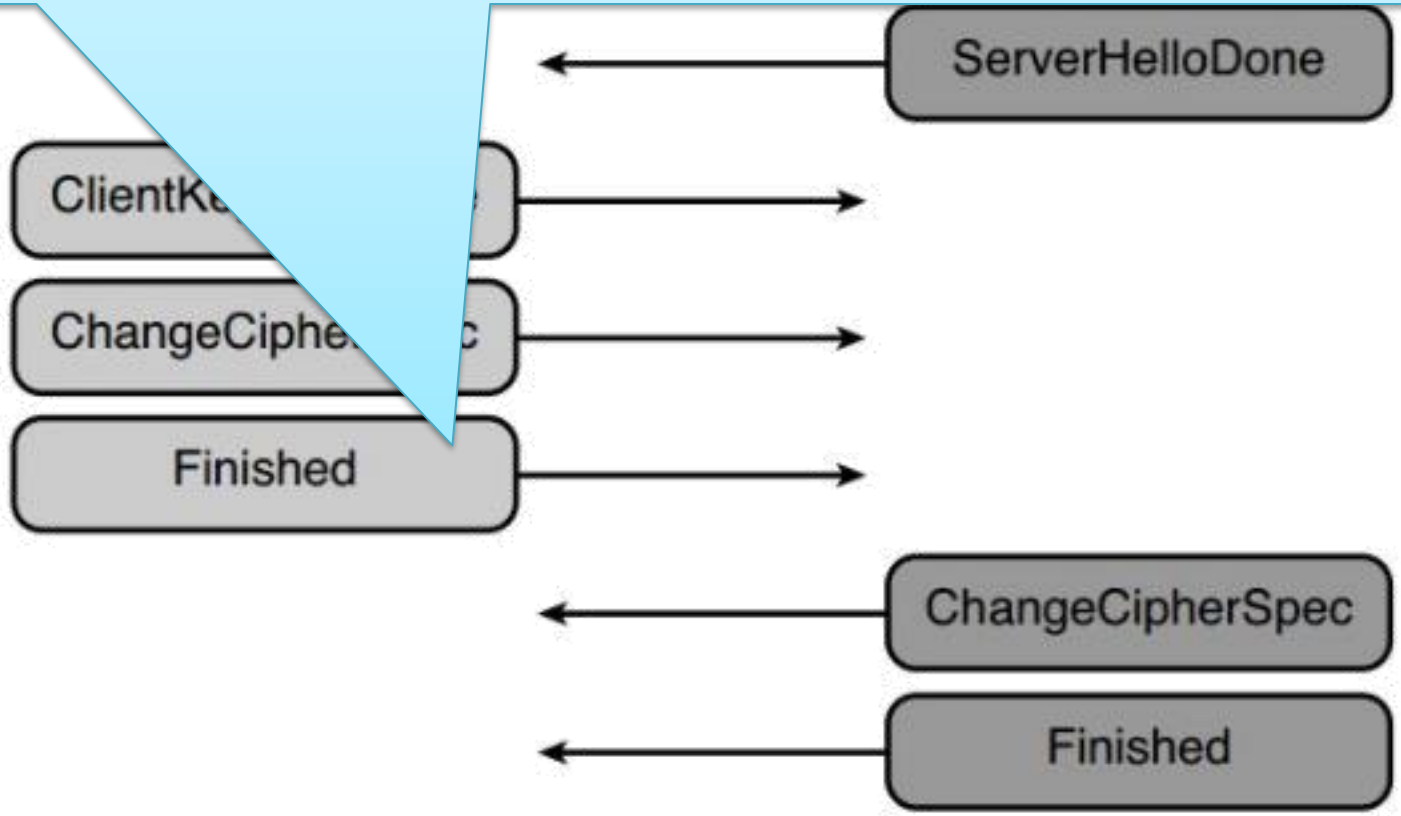
Giao thức Handshake

Thông báo: từ lúc này tất cả các gói tin trao đổi giữa client và server đều sẽ được **mã hóa** bằng các **thuật toán** và **session key** đã thương lượng



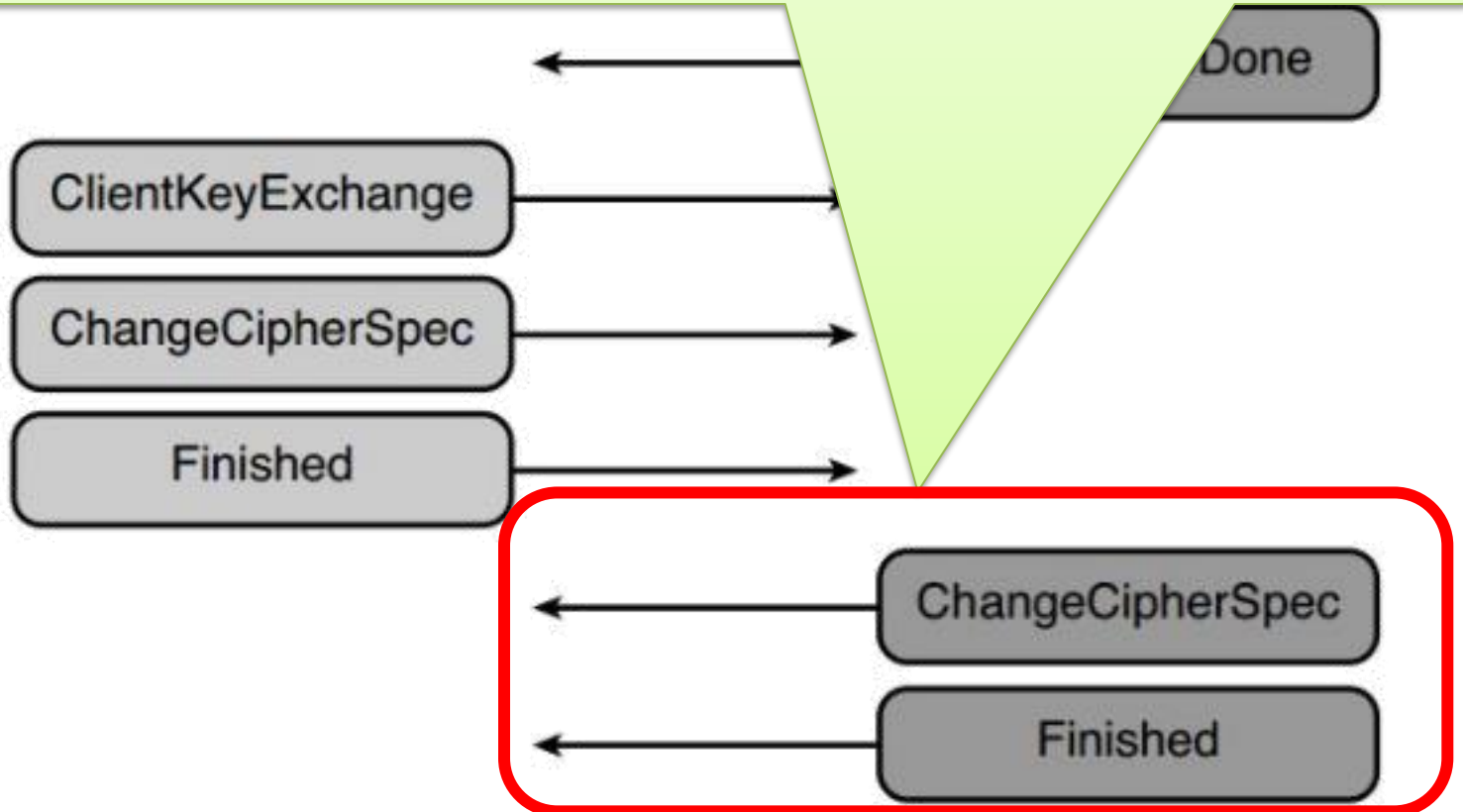
Giao thức Handshake

Client:
Gói Finished (được mã hóa) để kết thúc quá trình thiết lập SSL tunnel.



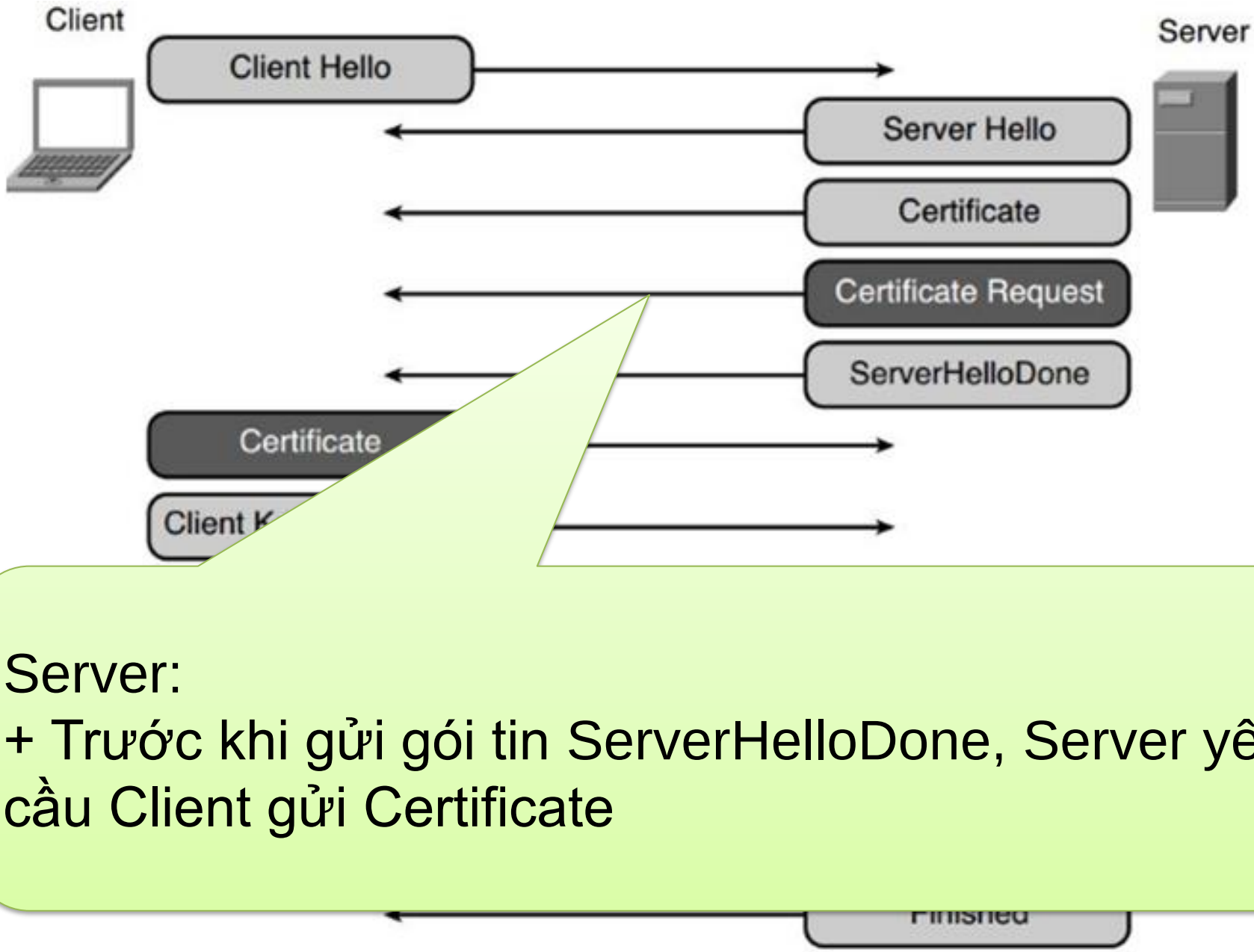
Giao thức Handshake

Server:
+ Gửi hai gói tin tương tự Client để thông báo: từ nay các gói tin sẽ được mã hóa
+ Kết thúc quá trình thiết lập SSL tunnel.



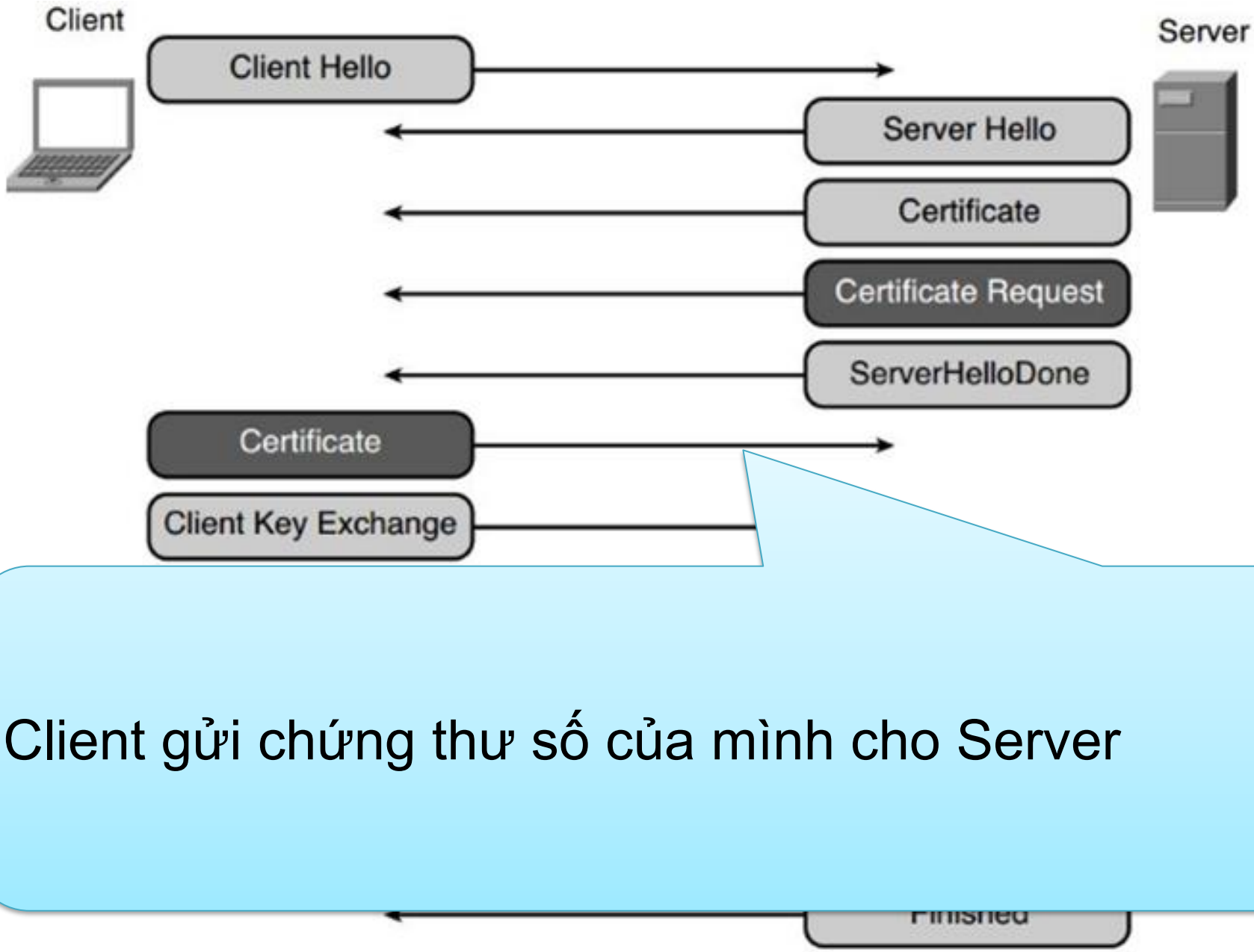
Server có thể yêu cầu xác thực Client

Trường hợp Server xác thực Client



Server:
+ Trước khi gửi gói tin ServerHelloDone, Server yêu cầu Client gửi Certificate

Trường hợp Server xác thực Client



Client gửi chứng thư số của mình cho Server

Trường hợp Server xác thực Client

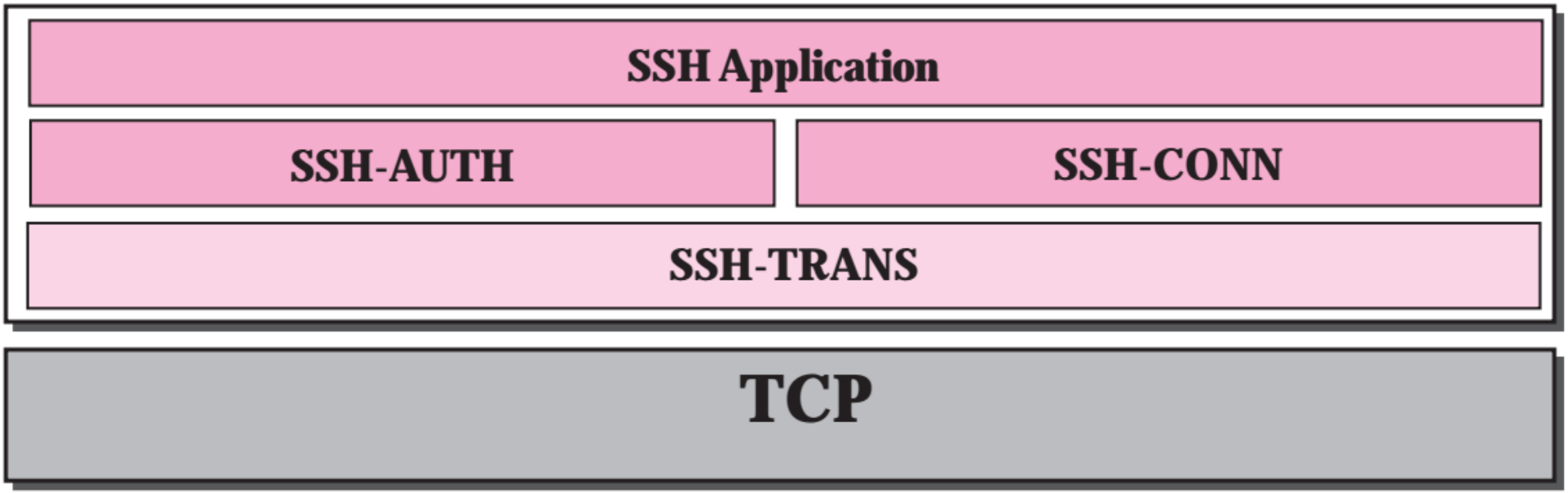
Server CertificateVerify:
+ H = Hash(tất cả các message đã trao đổi trước đó)
+ Vs = Kiểm tra chữ ký của Client lên H

Client CertificateVerify:
+ H = Hash(tất cả các message đã trao đổi trước đó)
+ Vc = Ký lên H

1 Bộ giao thức SSL/TLS

2 Bộ giao thức SSH

Kiến trúc giao thức SSH

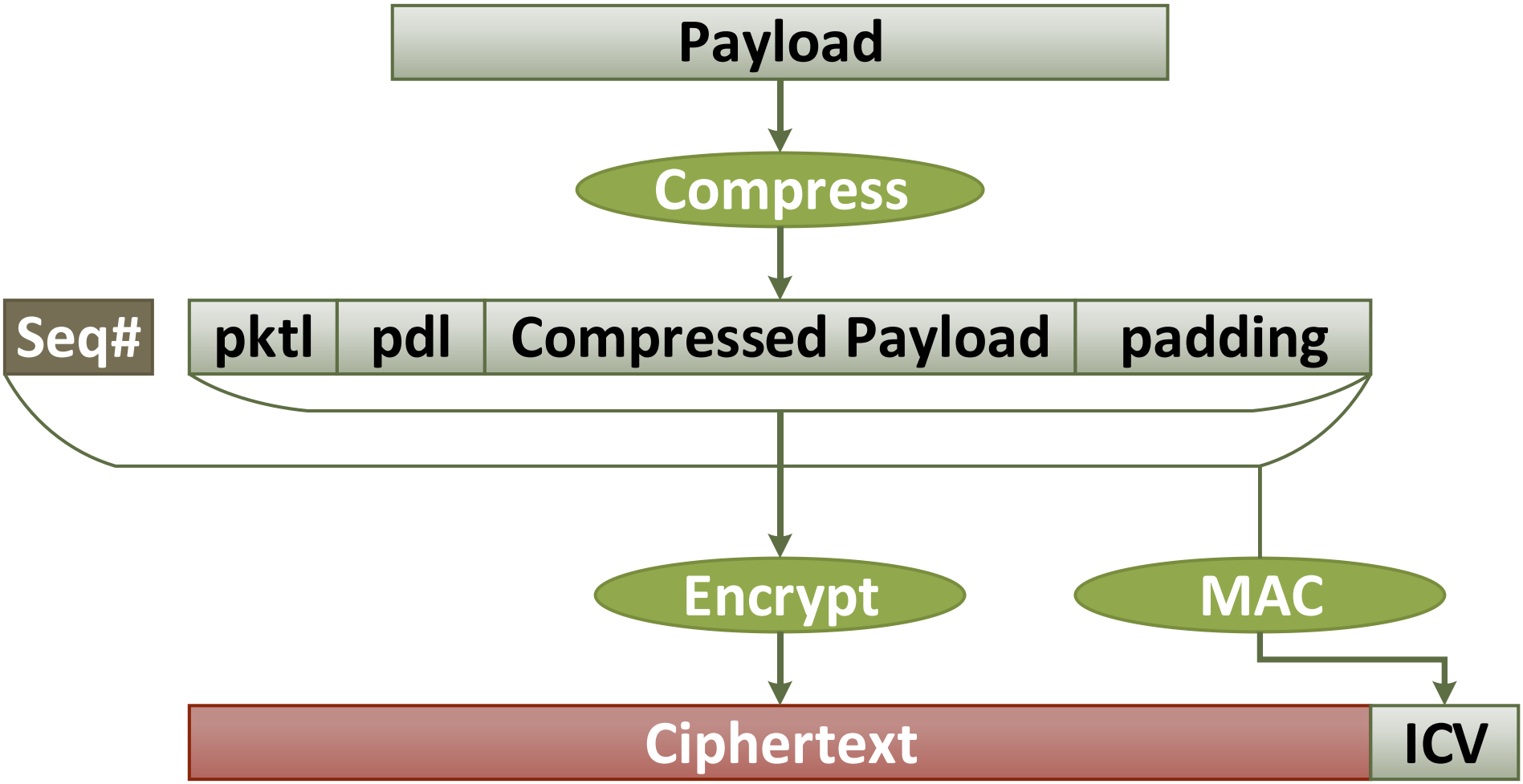


- Chức năng chủ yếu của SSH là remote login
- Ngoài ra còn có chức năng khác: truyền file an toàn với SCP, SFTP
- Nó có thể được sử dụng để bảo vệ bất kỳ ứng dụng mạng nào nhờ cơ chế port forwarding (tunneling)

1. Giao thức SSH-TRANS
2. Giao thức SSH-AUTH
3. Giao thức SSH-CONN
4. SSH Port Forwarding

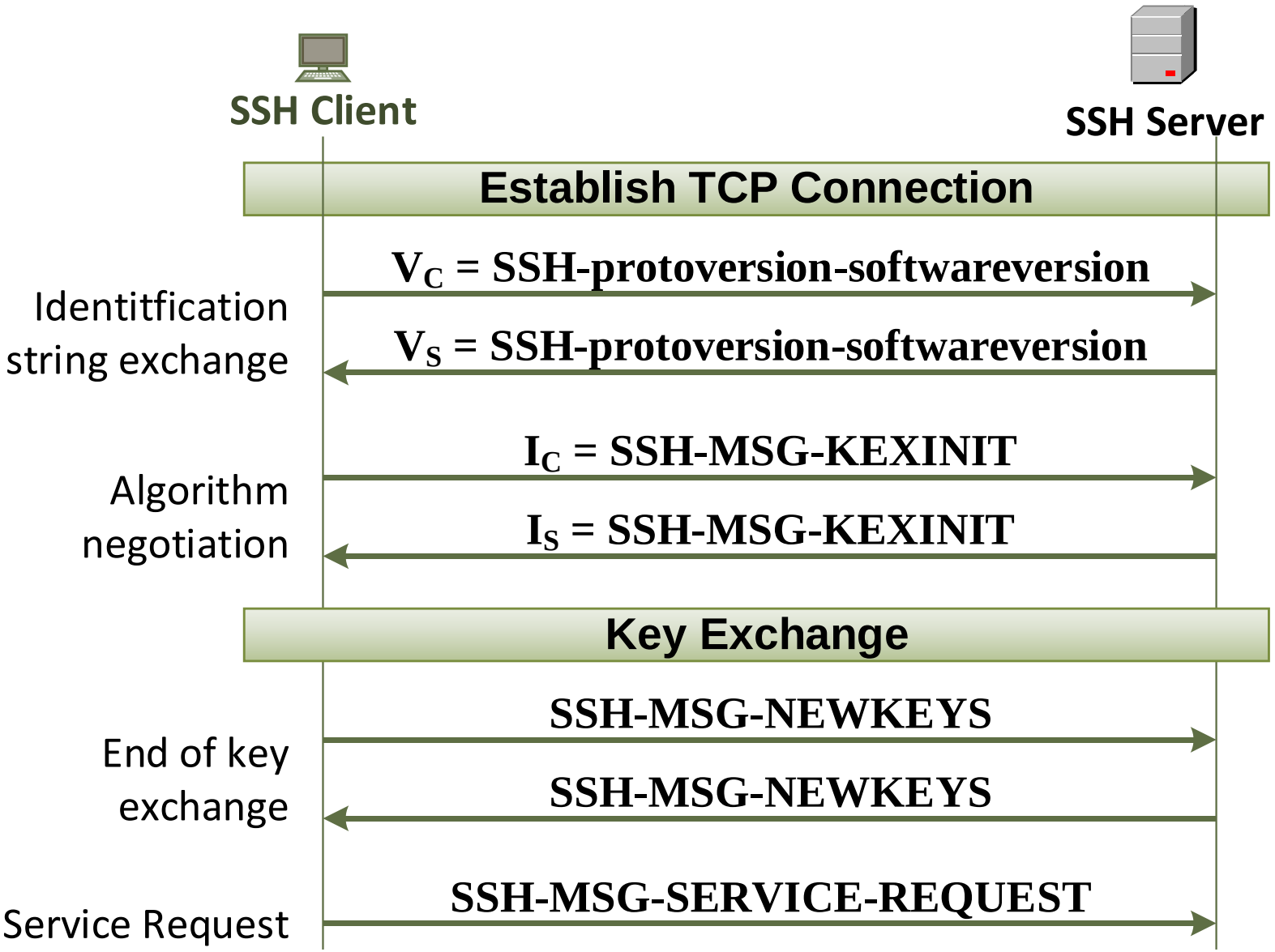
1. Giao thức SSH-TRANS
2. Giao thức SSH-AUTH
3. Giao thức SSH-CONN
4. SSH Port Forwarding

SSH-TRANS: SSH Packet

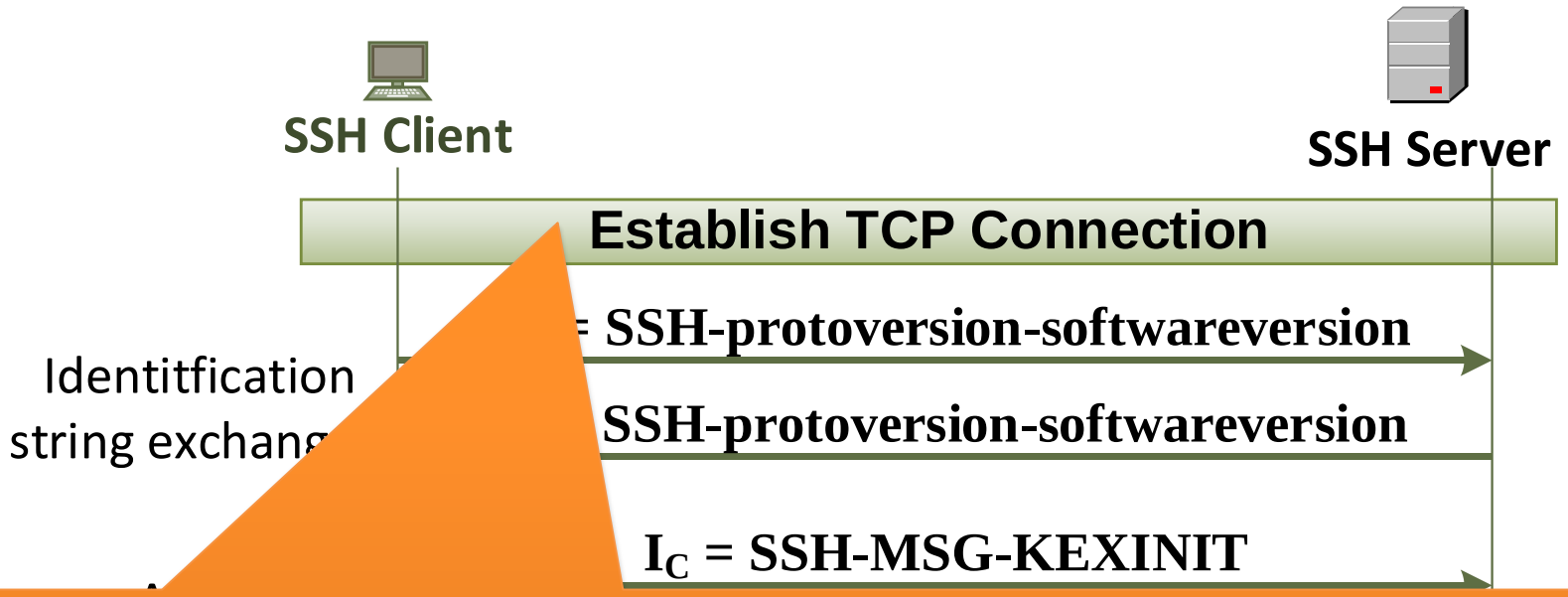


pktl = Packet Length
pdl = Padding Length

SSH-TRANS: Packet Exchanges

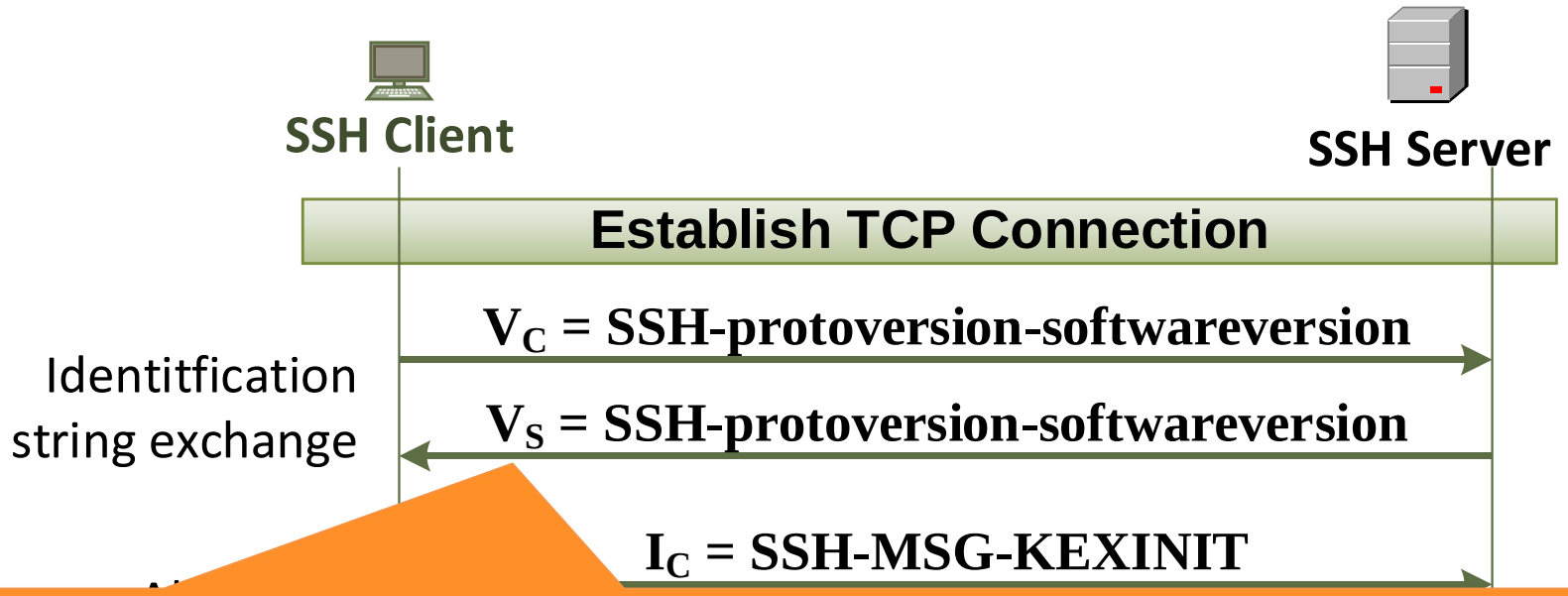


SSH-TRANS: Packet Exchanges



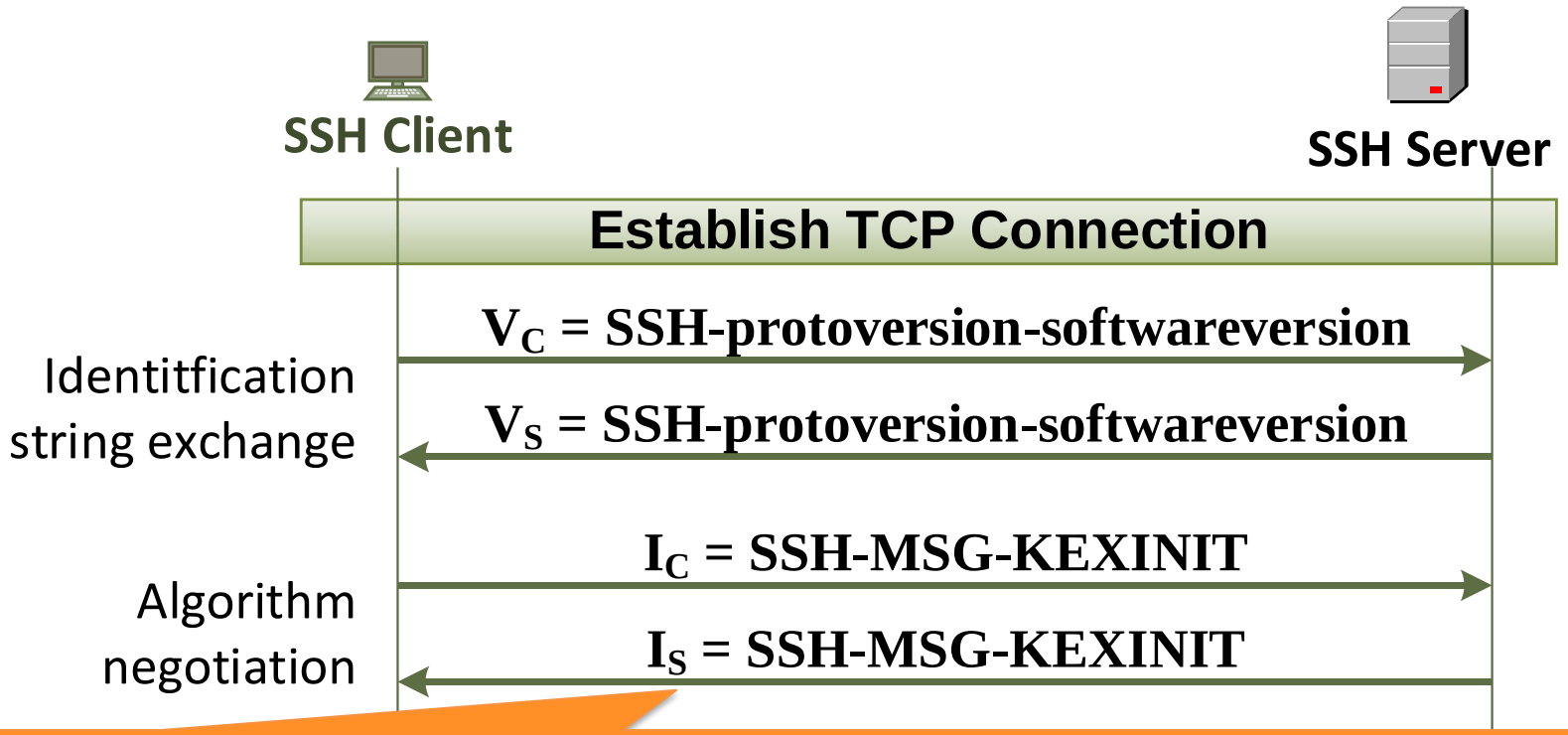
- Client mở kết nối TCP tới Server
- Không phải là một phần của SSH-TRANS

SSH-TRANS: Packet Exchanges



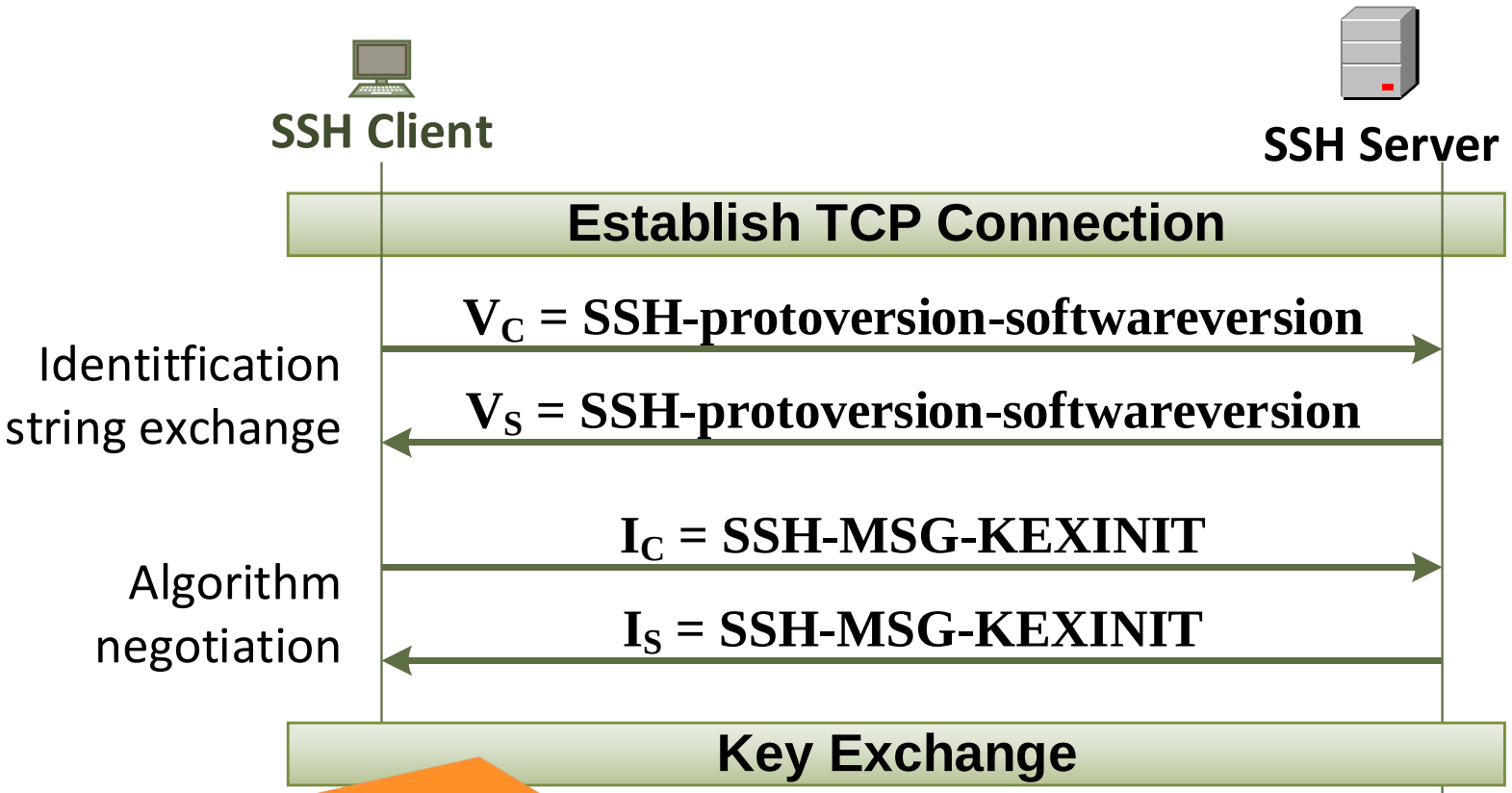
- Chuỗi định danh client, server
- Ví dụ: SSH-2.0-billsSSH_3.6.3q3<CR><LF>
- Được sử dụng trong xác thực server

SSH-TRANS: Packet Exchanges



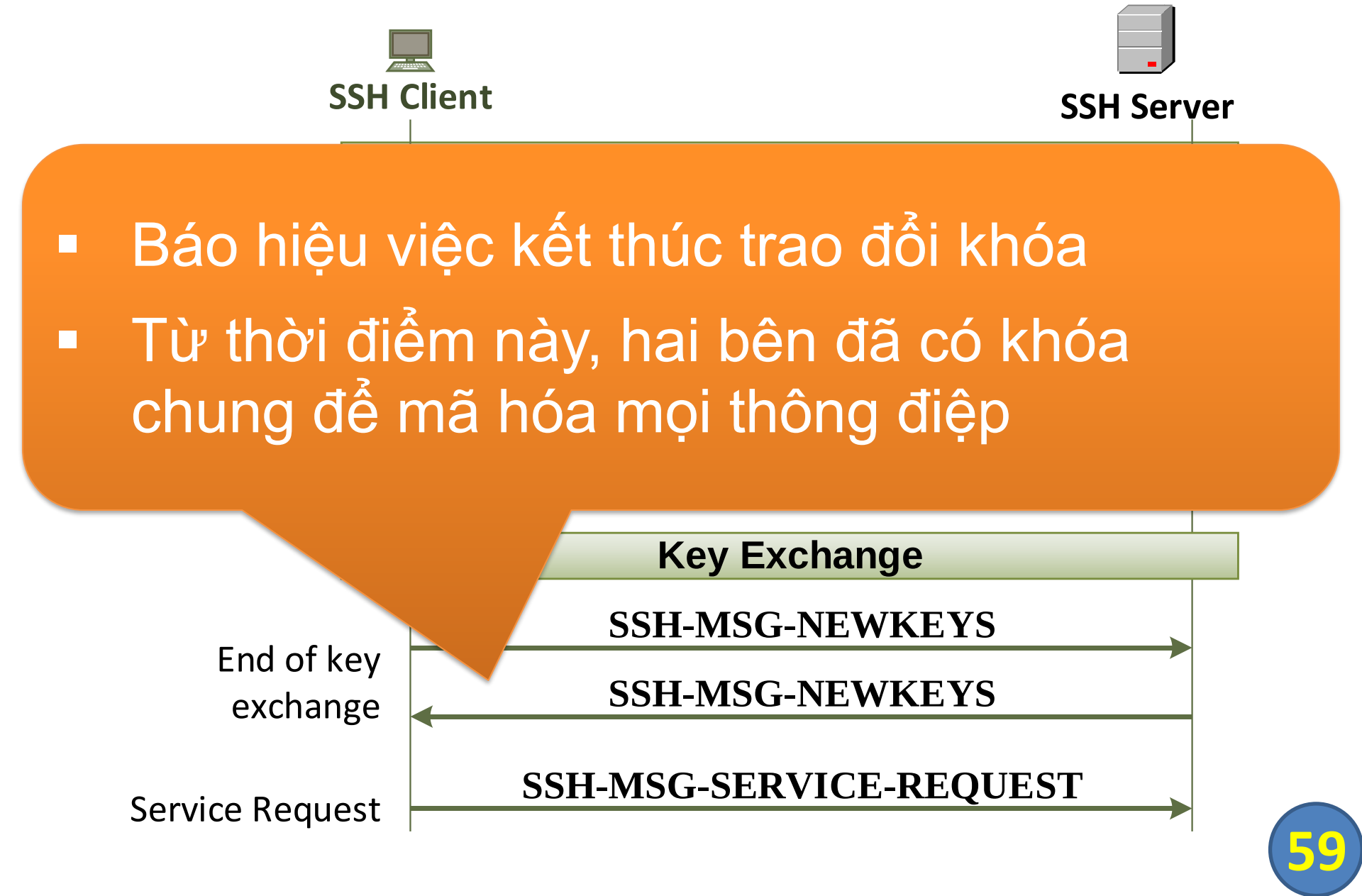
- Thỏa thuận bộ thuật toán và tham số: client đề xuất nhiều, server lựa chọn một bộ
- Bộ thuật toán: trao đổi khóa, mã hóa, xác thực, nén

SSH-TRANS: Packet Exchanges



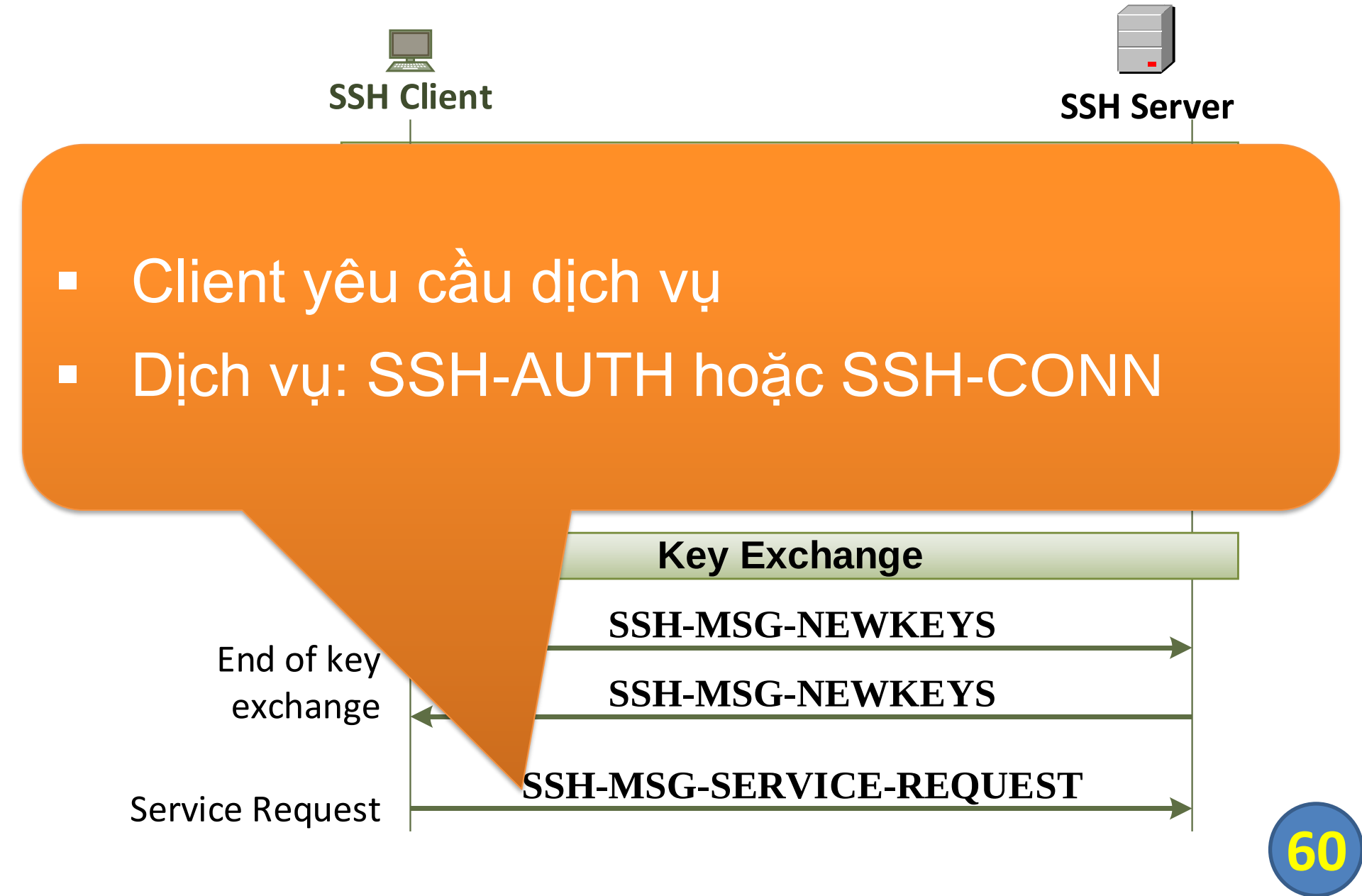
- Trao đổi khóa (Diffie-Hellman)
- Server được xác thực (bằng mật mã khóa công khai) trong quá trình trao đổi khóa

SSH-TRANS: Packet Exchanges



- Báo hiệu việc kết thúc trao đổi khóa
- Từ thời điểm này, hai bên đã có khóa chung để mã hóa mọi thông điệp

SSH-TRANS: Packet Exchanges



- Client yêu cầu dịch vụ
- Dịch vụ: SSH-AUTH hoặc SSH-CONN

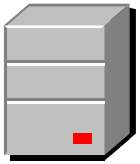
Server Authentication

- Sử dụng mật mã khóa công khai
  - Gọi khóa công khai là SSH Server Host Key (K_S)
  - Một server có thể có nhiều Host Key
  - Nhiều server có thể có chung một Host Key
- Mô hình tin cậy
  - Client quản lý danh sách các khóa công khai của các server tin cậy
  - Sử dụng PKI

Key Exchange and Server Authentication

SSH Client

K_S là một trong những khóa công khai của Server



SSH Server

e

//e = g^x mod p

K_S || f || s

//f = g^y mod p; K = e^y mod p;

//H = Hash(V_C || V_S || I_C || I_S || K_S || e || f || K);

//s = Sig_S(H)

//K = f^x mod p; Ver(K_S); Ver(s)

Dẫn xuất khóa và...

session_id:	H
Initial IV C-S:	HASH(K || H || "A" || session_id)
Initial IV S-C:	HASH(K || H || "B" || session_id)
Encryption key C-S:	HASH(K || H || "C" || session_id)
Encryption key S-C:	HASH(K || H || "D" || session_id)
Integrity key C-S:	HASH(K || H || "E" || session_id)
Integrity key S-C:	HASH(K || H || "F" || session_id)

1. Giao thức SSH-TRANS
2. Giao thức SSH-AUTH
3. Giao thức SSH-CONN
4. SSH Port Forwarding

SSH-AUTH

AuthenticationRequest

```
{
    byte      SSH_MSG_USERAUTH_REQUEST (50)
    string    user name
    string    service name
    string    method name
    ....      method-specific fields
}
```

SSH-AUTH

AuthenticationRequest

```
{
    byte      SSH_MSG_USERAUTH_REQUEST (50)
    string    user name
    string    service name
    string    method name
    ....      method-specific fields
}
```

Các kiểu dữ liệu: byte, uint32, string,... được định nghĩa tại Mục 5 của RFC 4251: SSH Protocol Architecture.

SSH-AUTH

AuthenticationRequest

```
{
    byte      SSH_MSG_USERAUTH_REQUEST (50)
    string    user name
    string    se ce name
    string    m  name
    ....      ecific fields
}
```

Định danh của client (người dùng)

SSH-AUTH

AuthenticationRequest

```
{
    byte      SSH_MSG_USERAUTH_REQUEST (50)
    string    user name
    string    service name
    string    method name
    ....      method specific fields
}
```

Dịch vụ muốn được thực thi sau khi xác thực thành công (thường là SSH Connection)

SSH-AUTH

AuthenticationRequest

```
{
    byte      SSH_MSG_USERAUTH_REQUEST (50)
    string    user name
    string    service name
    string    method name
    ....      method specific fields
}
```

- Phương thức xác thực được chọn (trong số các phương thức mà server hỗ trợ)
- "publickey", "password", "hostbased"
- "none" để yêu cầu danh sách

SSH-AUTH

AuthenticationRequest

```
{
    byte      SSH_MSG_USERAUTH_REQUEST (50)
    string    user name
    string    service name
    string    method name
    ....      method-specific fields
}
```

Tùy thuộc vào phương thức xác thực cụ thể được sử dụng

SSH-AUTH

AuthenticationFailure

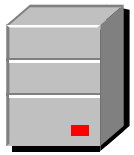
```
{
    byte        SSH_MSG_USERAUTH_FAILURE (51)
    name-list   authentications that can continue
    boolean     partial success
}
```

AuthenticationSuccess

```
{
    byte      SSH_MSG_USERAUTH_SUCCESS (52)
}
```

SSH-AUTH: Một kịch bản điển hình

SSH Client



SSH Server

AuthenticationRequest

//method = "none"

AuthenticationFailure

//name-list = danh sách các phương thức xác
//thực mà Server hỗ trợ

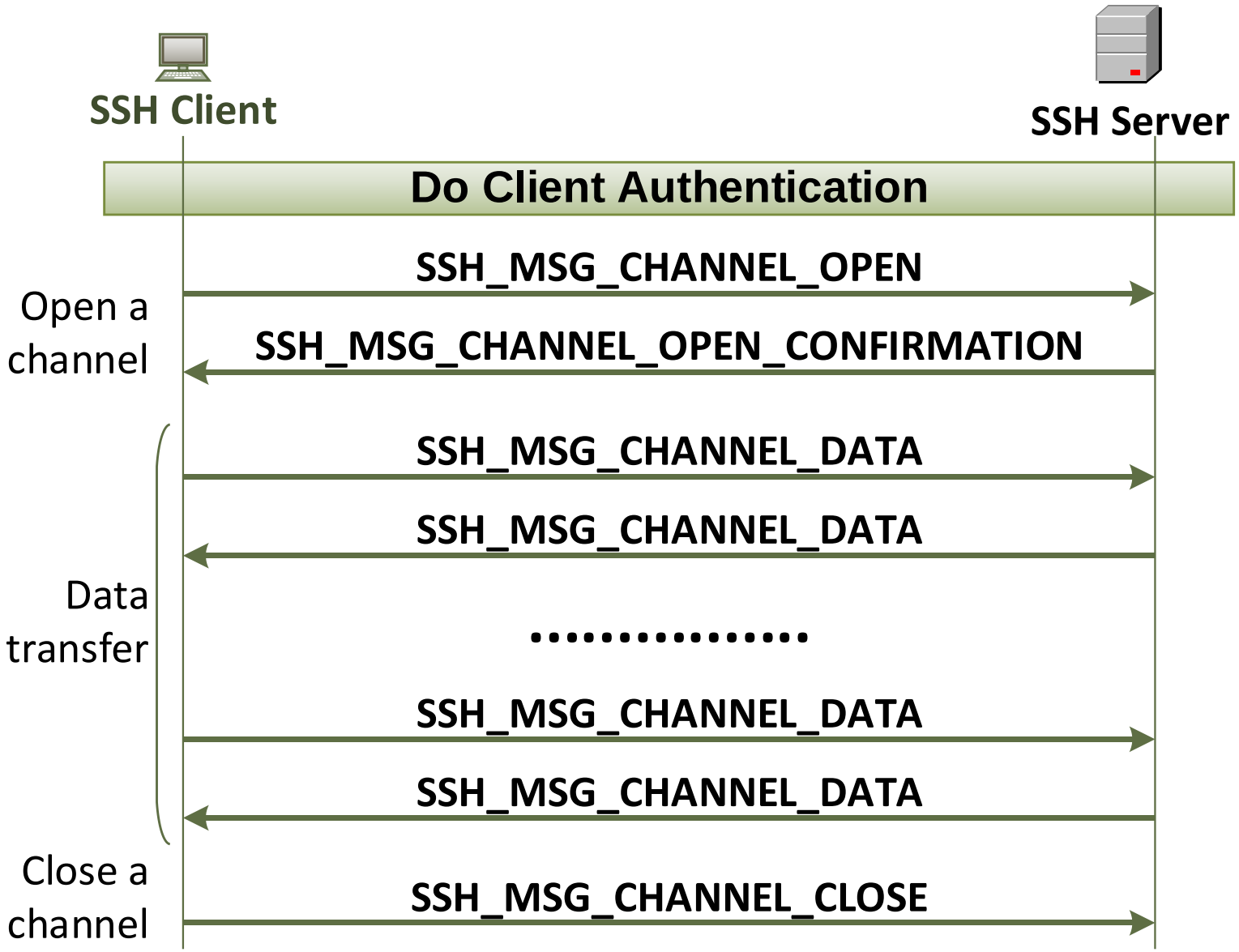
AuthenticationRequest

//method = tên phương thức cụ thể

AuthenticationSuccess

1. Giao thức SSH-TRANS
2. Giao thức SSH-AUTH
3. Giao thức SSH-CONN
4. SSH Port Forwarding

SSH-CONN: Kịch bản điển hình



SSH-CONN

ChannelOpen

```
{
    byte        SSH_MSG_CHANNEL_OPEN (90)
    string      channel type
    uint32      sender channel
    uint32      initial window size
    uint32      maximum packet size
    ....        channel type specific data follow
}
```

SSH-CONN

ChannelOpen

```
{
    byte        SSH_MSG_CHANNEL_OPEN (90)
    string      channel type
    uint32      se    er channel
    uint32      ir    indow size
    uint32              packet size
    ....                specific data follow
}
```

- "session": thực thi ứng dụng từ xa
- "x11": X Window System
- "forwarded-tcpip": Remote Port Forwarding
- "direct-tcpip": Local Port Forwarding

SSH-CONN

ChannelOpen

```
{
    byte        SSH_MSG_CHANNEL_OPEN (90)
    string      channel type
    uint32      sender channel
    uint32      initi  window size
    uint32      m      packet size
    ....                specific data follow
}
```

Số hiệu kênh đối với **sender** của thông điệp này.

Hai bên có thể sử dụng hai số hiệu khác nhau để tham chiếu đến cùng một kênh.

SSH-CONN

ChannelOpen

```
{
    byte        SSH_MSG_CHANNEL_OPEN (90)
    string      channel type
    uint32      sender channel
    uint32      initial window size
    uint32      ma      m packet size
    ....            specific data follow
}
```

- Số byte tối đa được phép gửi cho **sender** của thông điệp này trước khi phải chờ đợi nó (sender của thông điệp này) gửi thông điệp điều chỉnh kích thước cửa sổ
- Mỗi bên tự quyết định giá trị này

SSH-CONN

ChannelOpenConfirmation

```
{
    byte        SSH_MSG_CHANNEL_OPEN_CONFIRMATION(91)
    uint32      recipient channel
    uint32      sender channel
    uint32      initial window size
    uint32      maximum packet size
    ....        channel type specific data follows
}
```

SSH-CONN

ChannelOpenConfirmation

```
{
    byte        SSH_MSG_CHANNEL_OPEN_CONFIRMATION(91)
    uint32      recipient channel
    uint32      send  channel
    uint32      init  dow size
    uint32      m     acket size
    ....              specific data follows
}
```

Số hiệu kênh đối với **receiver** của thông điệp này (tức là **sender** của thông điệp ChannelOpen)

SSH-CONN

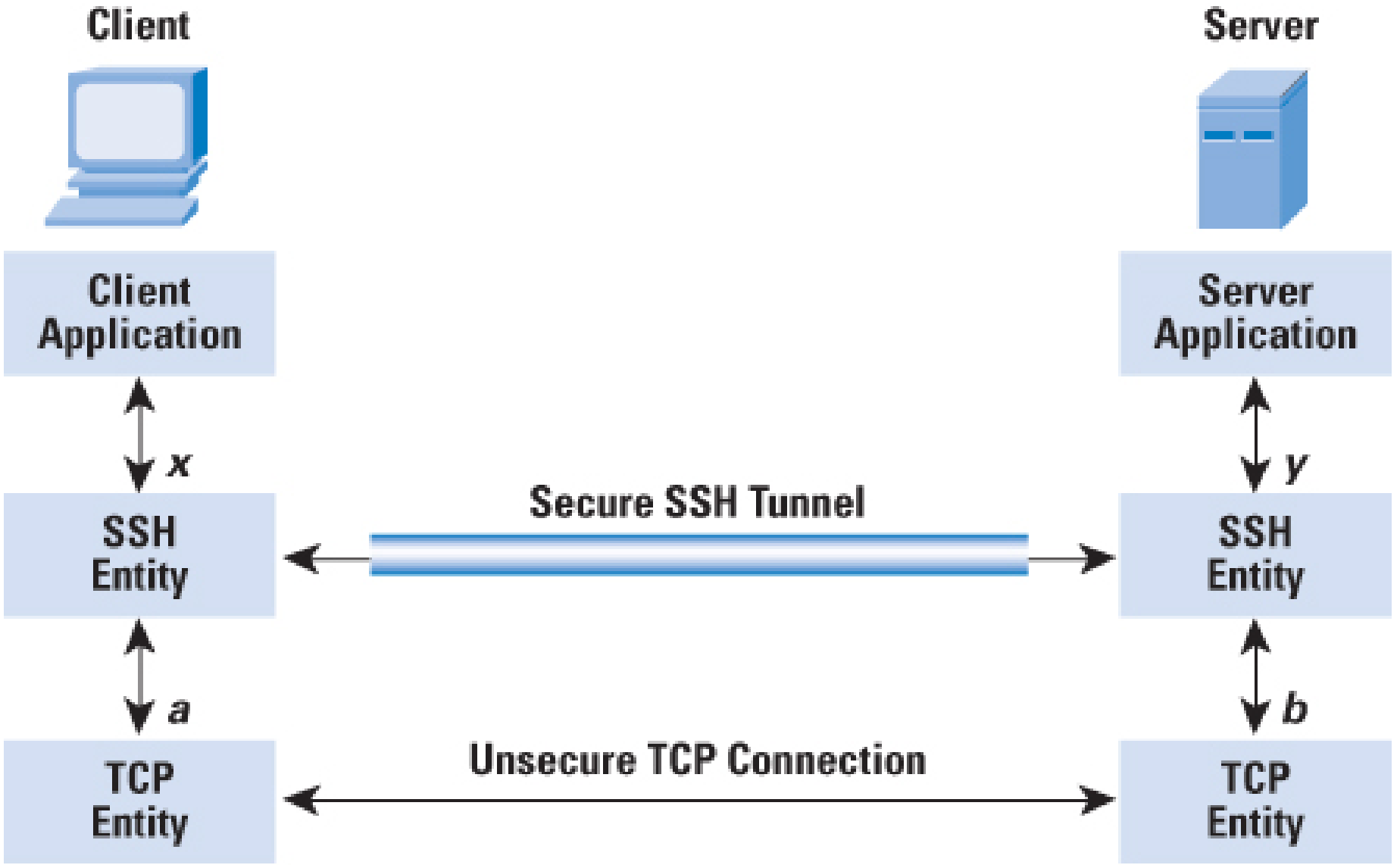
ChannelOpenConfirmation

```
{
    byte        SSH_MSG_CHANNEL_OPEN_CONFIRMATION(91)
    uint32      recipient channel
    uint32      sender channel
    uint32      initi  window size
    uint32      m      packet size
    ....                specific data follows
}
```

Số hiệu kênh đối với **sender** của thông điệp này

1. Giao thức SSH-TRANS
2. Giao thức SSH-AUTH
3. Giao thức SSH-CONN
4. SSH Port Forwarding

Ý tưởng về Port Forwarding



SSH Port Forwarding

Có 3 loại SSH port forwarding là:

- *Local port forwarding:* là dạng kết nối từ phía SSH client được chuyển tiếp qua SSH server, rồi đi đến host/server đích.
- *Remote port forwarding:* kết nối từ phía SSH server được chuyển tiếp qua SSH client, rồi đi đến host/server đích.
- *Dynamic port forwarding:* tương tự "local port forwarding", kết nối từ phía SSH client được chuyển tiếp qua SSH server, rồi đến đích tuỳ ý không định trước.

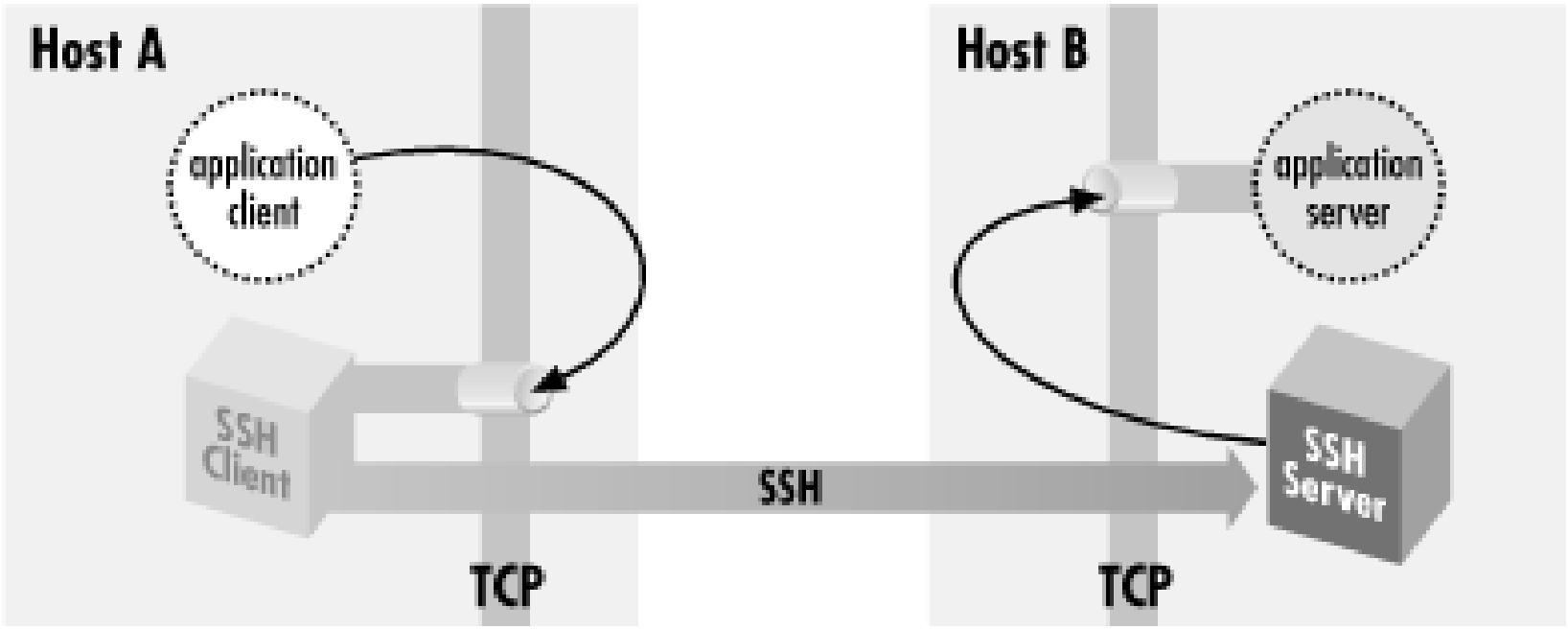
Local Port Forwarding

Mở kết nối ssh tới ssh server với tuỳ chọn "-L port:*host:hostport*" trong đó

port là cổng ở phía ssh client được chỉ định để mở socket.

host:hostport là socket đích muốn tới (nhìn từ phía ssh server).

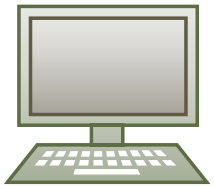
Local Port Forwarding



$ ssh –L<lport>:localhost:<rport> remote.net

- Application Server nằm cùng máy với SSH Server
- Application Client nằm cùng máy với SSH Client
- Cho phép Application Client kết nối với Application Server thông qua cổng <lport> trên máy cục bộ (SSH client).

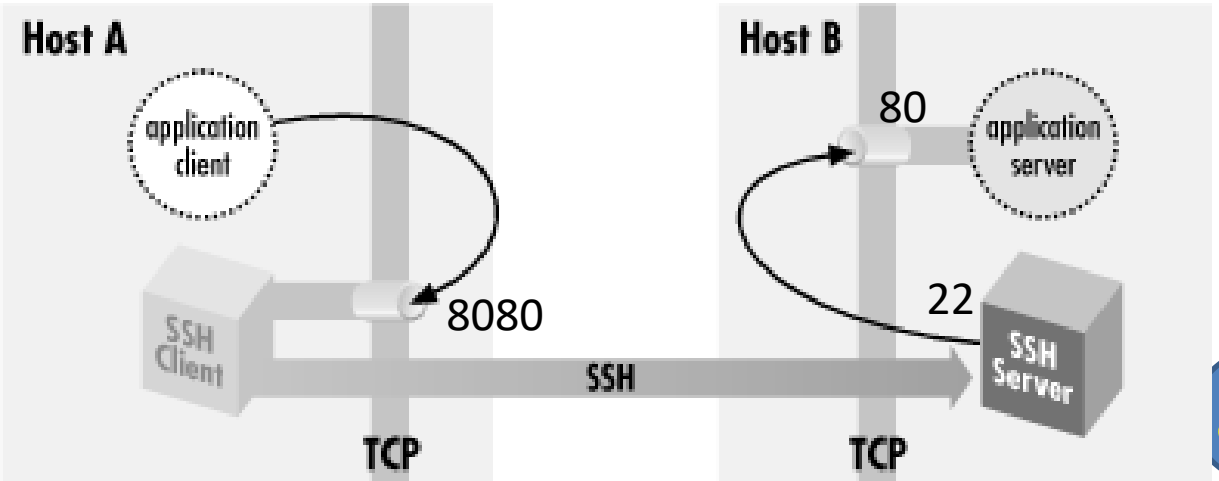
Local Port Forwarding



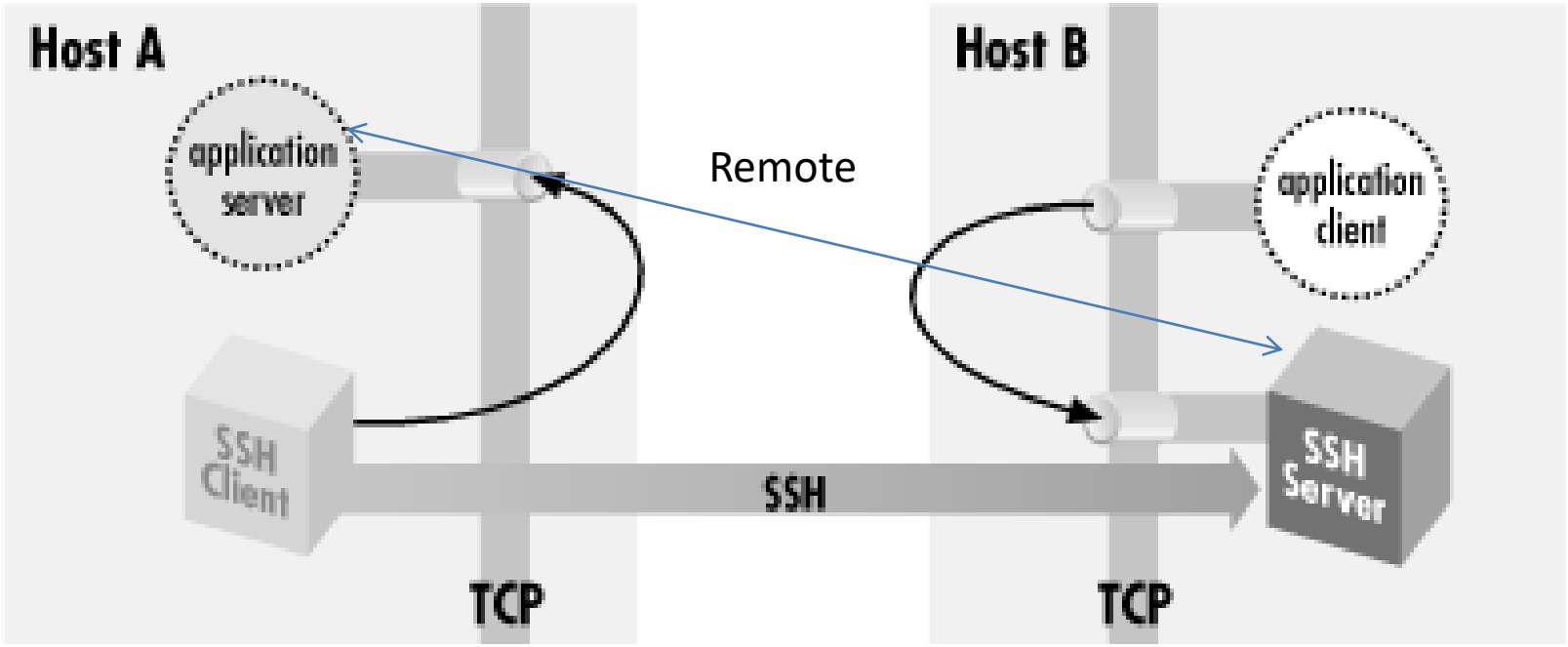
IP: 10.0.0.123
OS: Ubuntu
SSH Client
Web Browser

IP: 10.0.0.1
OS: Ubuntu
SSH Server (OpenSSH, 22)
Web Server (Apache2, 80)

ssh -L 8080:**10.0.0.1**:80 **10.0.0.1**
http://localhost:8080



Remote Port Forwarding



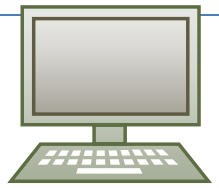
$ ssh –R<rport>:localhost:<lport> remote.net

- Application Server nằm cùng máy với SSH Client
- Application Client nằm cùng máy với SSH Server
- Cho phép Application Client kết nối với Application Server thông qua cổng <rport> trên máy cục bộ (SSH Server)

Remote Port Forwarding

Home Comp

IP: 10.0.0.21
SSH Server
Web Browser



Work Comp

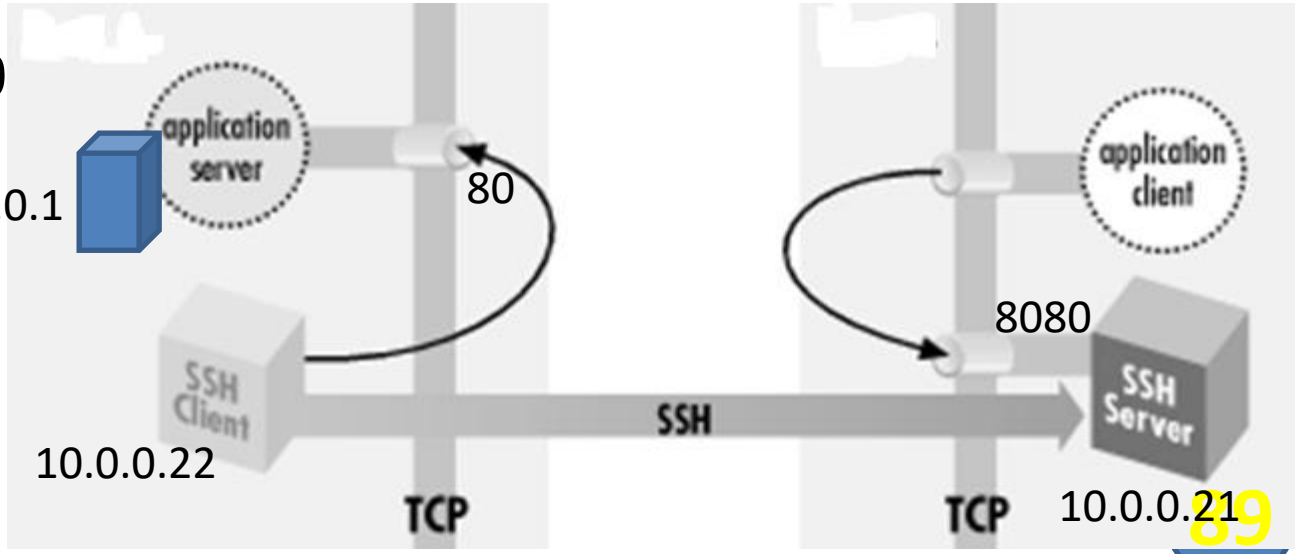
IP: 10.0.0.22
SSH Client

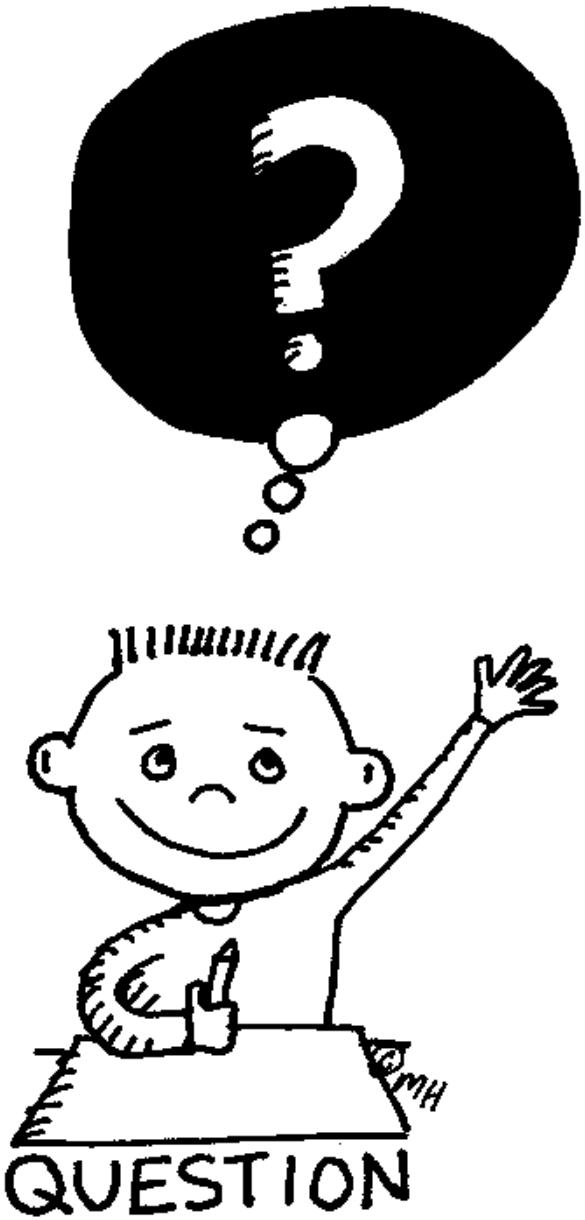
Work Server

IP: 10.0.0.1
Web Server

ssh -R 8080:**10.0.0.1**:80 **10.0.0.21**

http://localhost:8080




QUESTION

Thực hành

- Bài tập đã giao
- Dùng wireshare để theo dõi một kết nối tới website bất kỳ có sử dụng HTTPS
- Thử nghiệm Port Forwarding với SSH