Network Security (NetSec)

Summer 2015
Chapter 04: Transport Level Security
Module 02: Secure Socket Layers (SSL)

Prof. Dr.-Ing. Matthias Hollick

Technische Universität Darmstadt
Secure Mobile Networking Lab - SEEMOO
Department of Computer Science
Center for Advanced Security Research Darmstadt - CASED

Mornewegstr. 32
D-64293 Darmstadt, Germany
Tel.+49 6151 16-70922, Fax. +49 6151 16-70921
http://seemoo.de or http://www.seemoo.tu-darmstadt.de

Prof. Dr.-Ing. Matthias Hollick
matthias.hollick@seemoo.tu-darmstadt.de

Learning Objectives

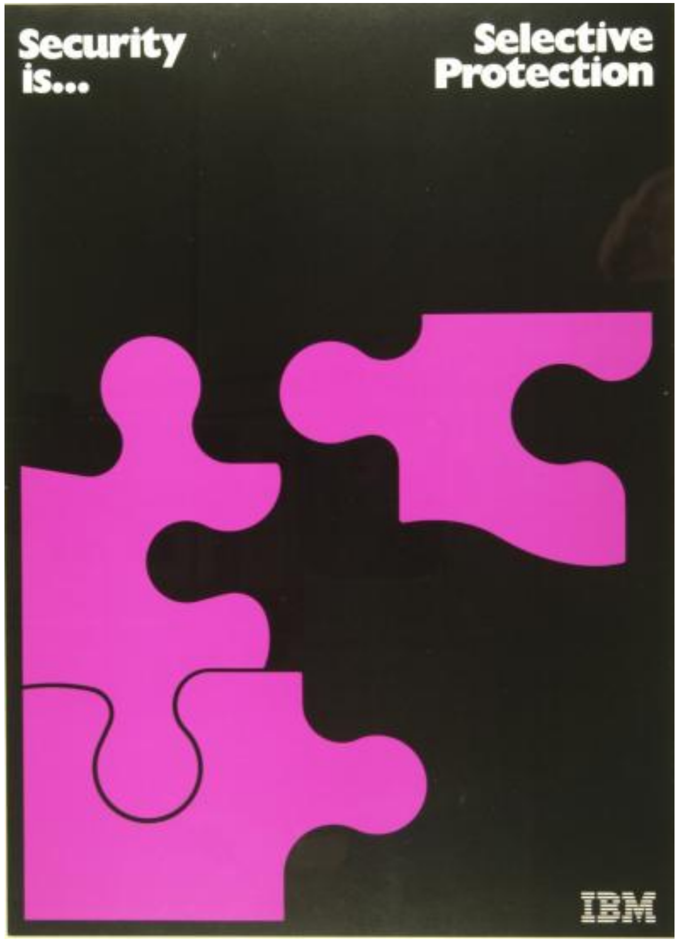
Security objectives, mechanisms and limitations on transport layer (or between network layer and application layer)

- Identify the scope of protection as well as the trade-offs involved in securing networks on transport layer
- Understand the fundamental design principles of transport layer security protocols
- Discuss toy and real-world protocols to secure the transport layer
  - The Secure Socket Layer protocol

- In preceding module
  - A toy SSL protocol
- In subsequent modules
  - TLS (transport layer security) and SSH (secure shell)

Overview of this Module

(1) SSL incredients
(2) SSL architecture
(3) SSL record protocol
(4) SSL handshake
(5) Recommended readings



Source: thesilverliningblog.com

Chapter 04, Module 02

SSL Security Services

Peer entity authentication:

- Prior to any communications between a client and a server, an authentication protocol is run to authenticate the peer entities
- Upon successful completion of the authentication dialogue an SSL session is established between the peer entities

User data confidentiality:

- If negotiated upon session establishment, user data is encrypted
- Different encryption algorithms can be negotiated: RC4, DES, 3DES, IDEA, AES, … can be extended to accommodate other ciphe rsuites

User data integrity:

- A MAC based on a cryptographic hash function is appended to user data
- The MAC is computed with a negotiated secret in prefix-suffix mode
- Either MD5 or SHA can be negotiated for MAC computation

SSL Session & Connection State

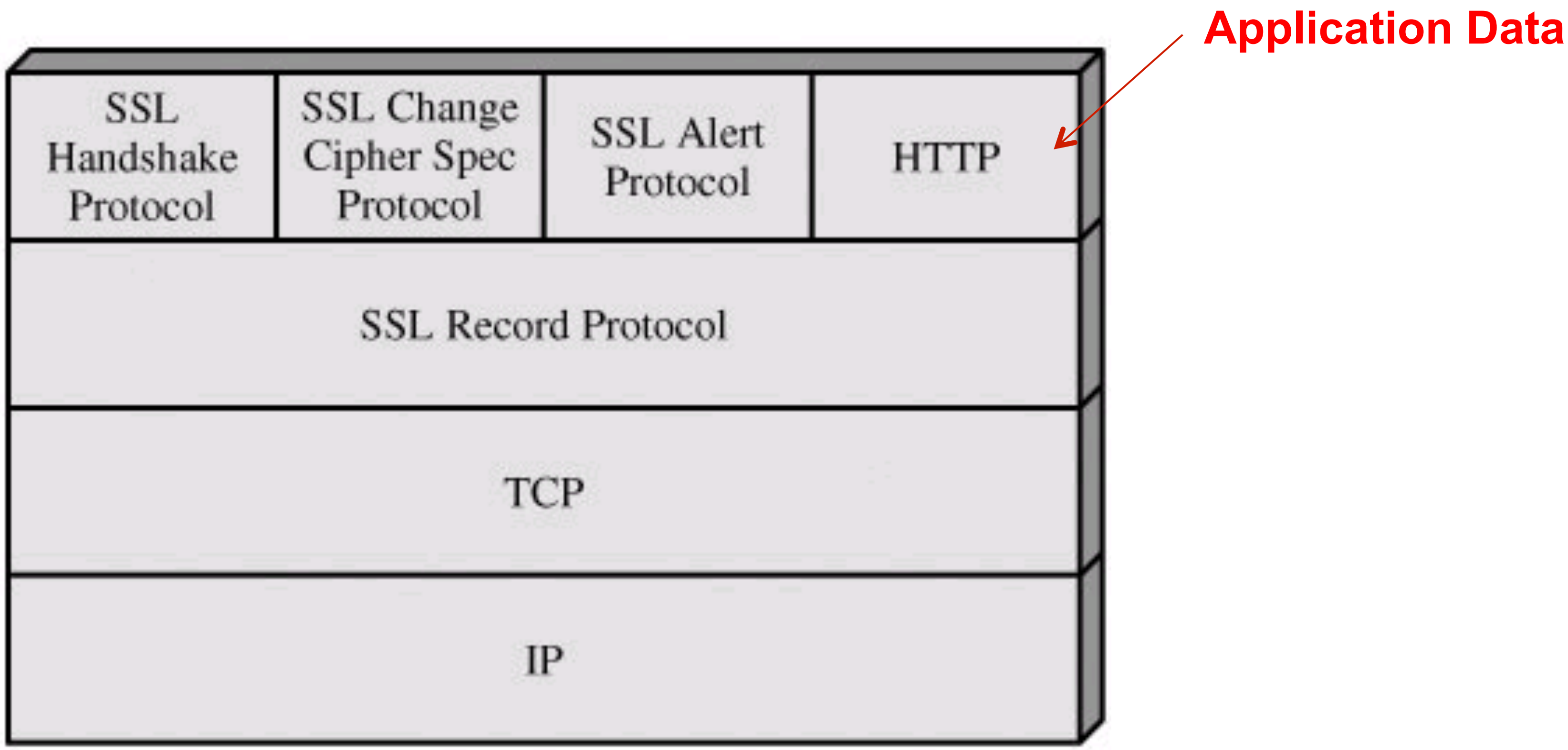
Session state:

- *Session identifier:* a byte sequence chosen by the server
- *Peer certificate:* X.509 v.3 certificate of the peer (optional)
- *Compression method:* algorithm to compress data prior to encryption
- *Cipher spec:* specifies cryptographic algorithms and parameters
- *Master secret:* a negotiated shared secret of length 48 byte
- *Is resumable:* a flag indicating if the session supports new connections

Connection state:

- *Server and client random:* byte sequences chosen by server and client
- *Server write MAC secret:* used in MAC computations by the server
- *Client write MAC secret:* used in MAC computations by the client
- *Server write key:* used for encryption by server and decryption by client
- *Client write key:* used for encryption by client and decryption by server

SSL Protocol Stack



Content types in record header

change_cipher_spec (20)

- indicates change in encryption and authentication algorithms

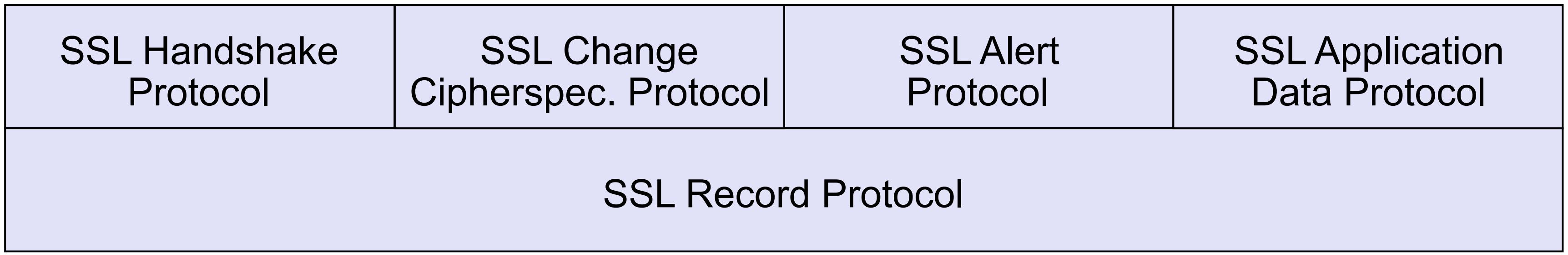
alert (21)

- signaling errors during handshake (or closure)

handshake (22)

- initial handshake messages are carried in records of type "handshake"
- Hankshake messages in turn have their own "sub" types

application_data (23)



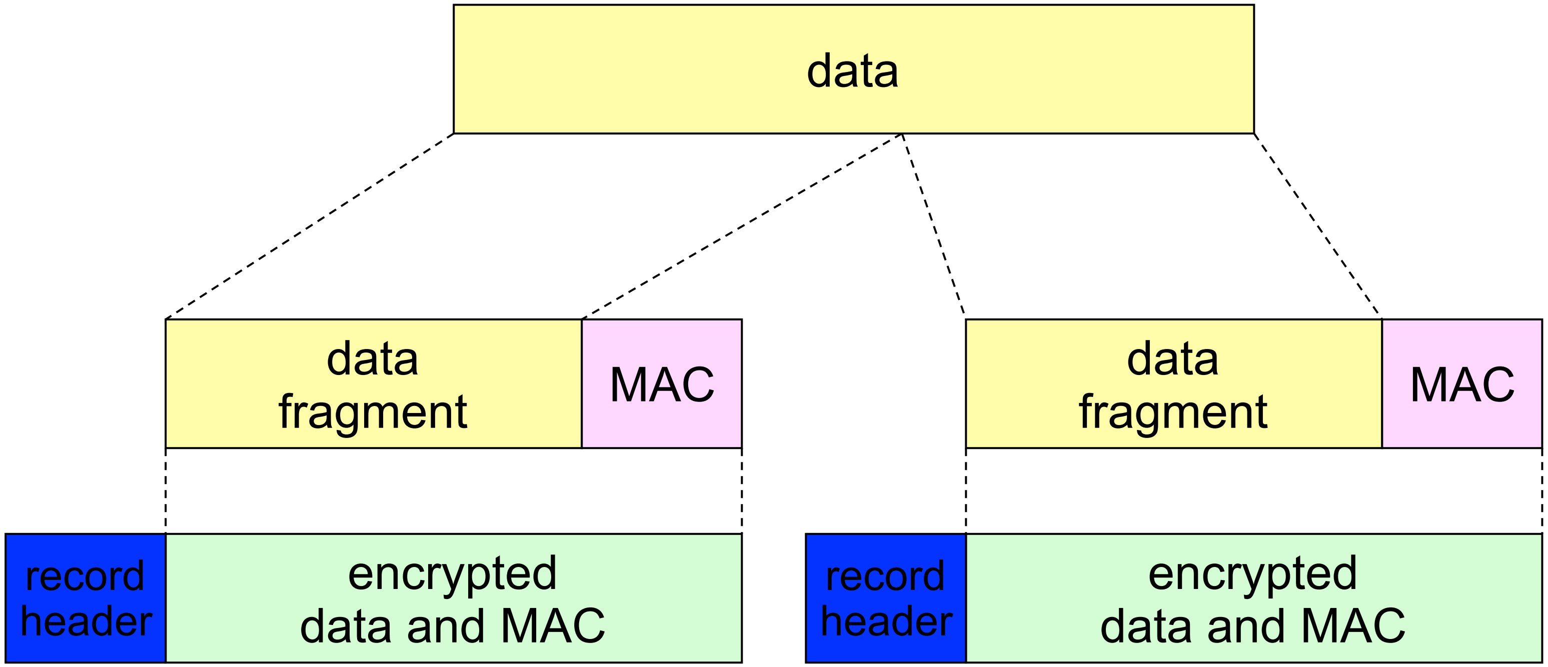
SSL Record Protocol

record header: content type; version; length

MAC: includes sequence number, MAC key M_x

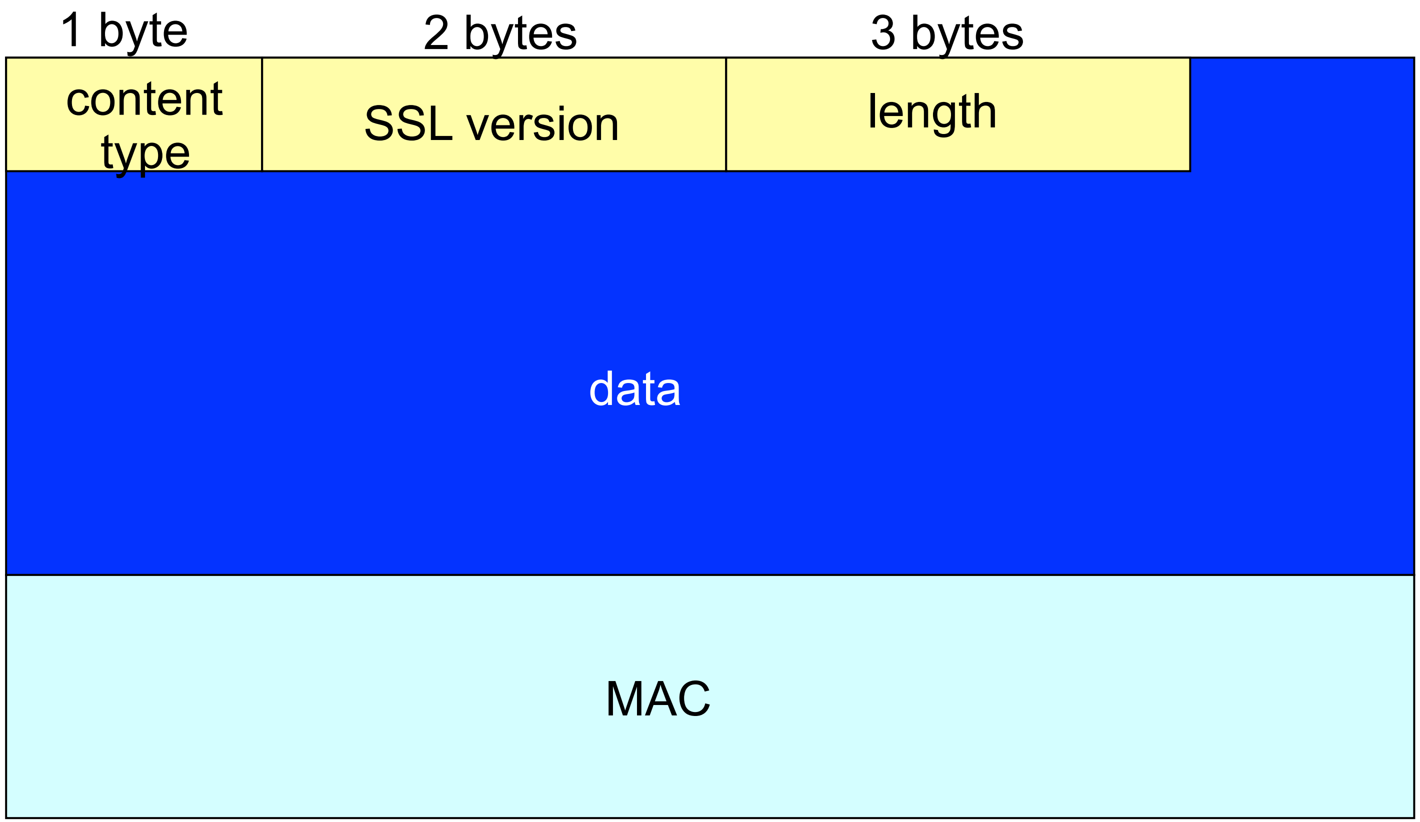
Fragment: each SSL fragment 2^{14} bytes (~16 Kbytes)

Compression (optional) of plaintext records



SSL Record Format

Data and MAC encrypted (symmetric algo)



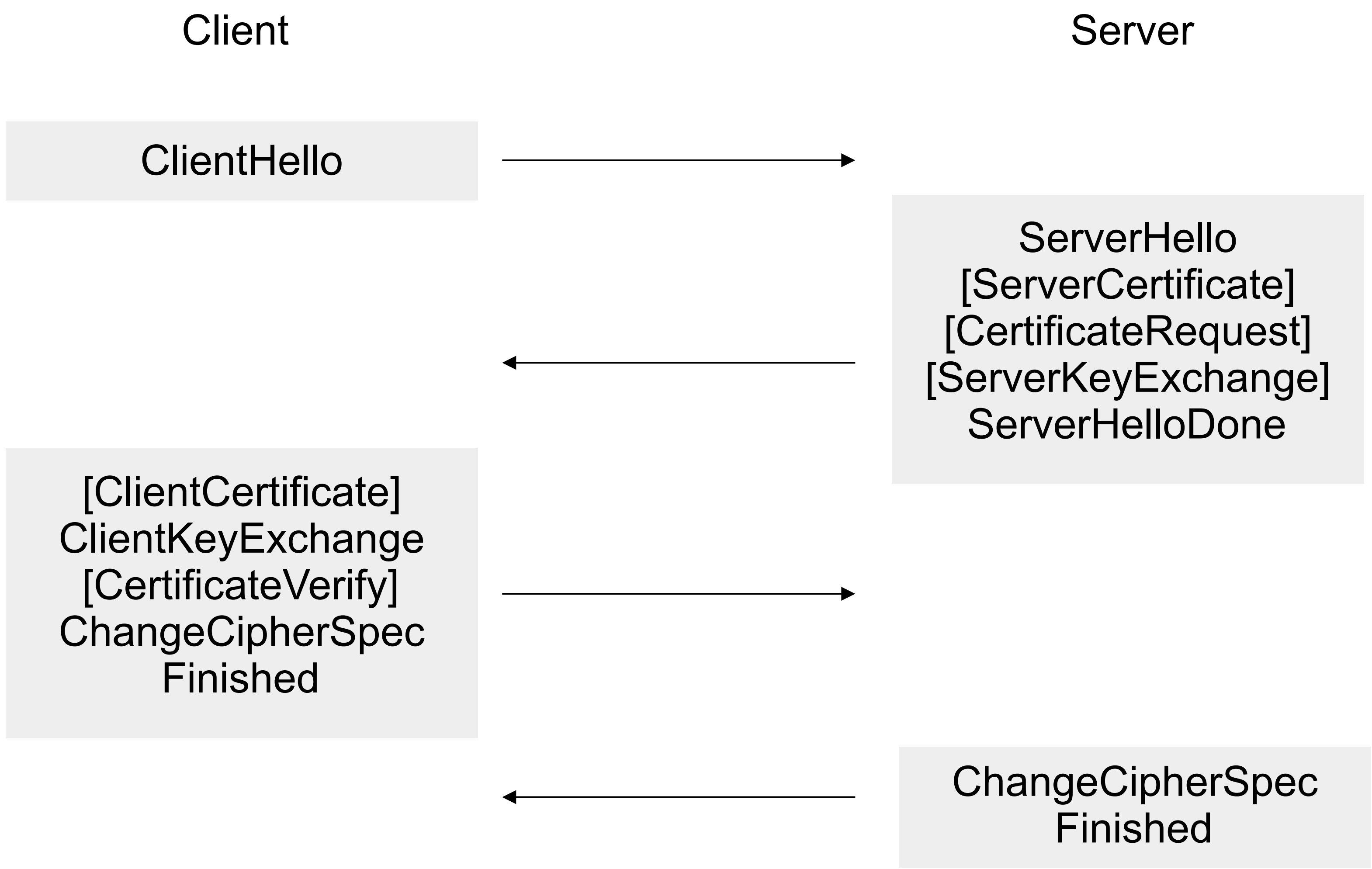
SSL: Handshake (1)

Purpose

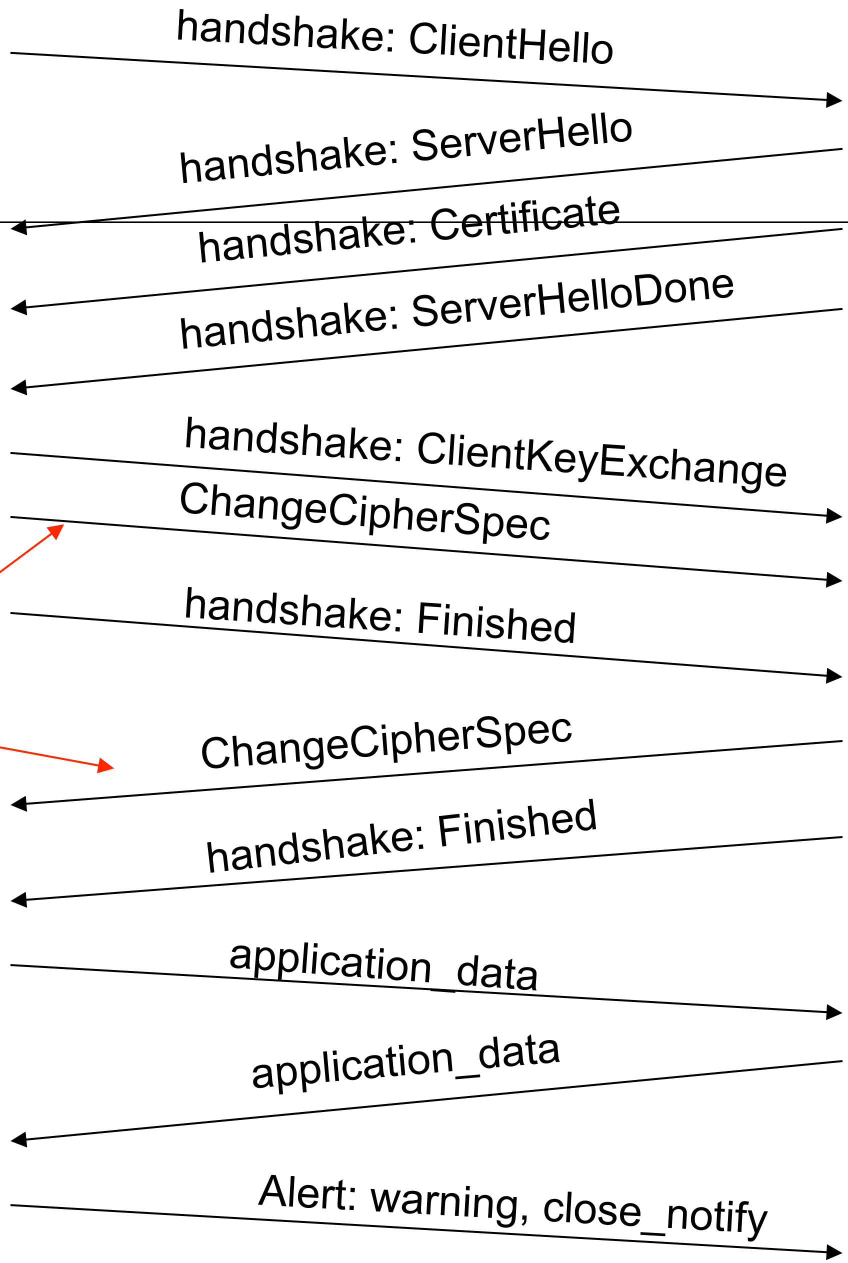
1. Server authentication
2. Negotiation: agree on crypto algorithms
3. Establish keys
4. Client authentication (optional)

- An SSL session can be negotiated to be resumable:
  - Resuming and duplicating SSL sessions (i.e. creating a new connection) allows to re-use established security context
  - This is very important for securing HTTP traffic, as usually every item on a web page may be transferred over an individual TCP connection
  - When resuming / duplicating an existing session, an abbreviated handshake is performed

SSL Handshake Protocol: Full Handshake



Again: Handshake



Everything henceforth is encrypted

TCP Fin follow

SSL: Handshake (contd.)

1. Client sends list of algorithms it supports, along with client nonce
2. Server chooses algorithms from list; sends back: choice + certificate + server nonce
3. Client verifies certificate, extracts server's public key, generates pre_master_secret, encrypts with server's public key, sends to server
4. Client and server independently compute encryption and MAC keys from pre_master_secret and nonces
5. Client sends a MAC of all the handshake messages
6. Server sends a MAC of all the handshake messages

Why would you add steps 5 and 6?

SSL: Handshake (contd.)

Last 2 steps protect handshake from tampering

Client typically offers range of algorithms, some strong, some weak

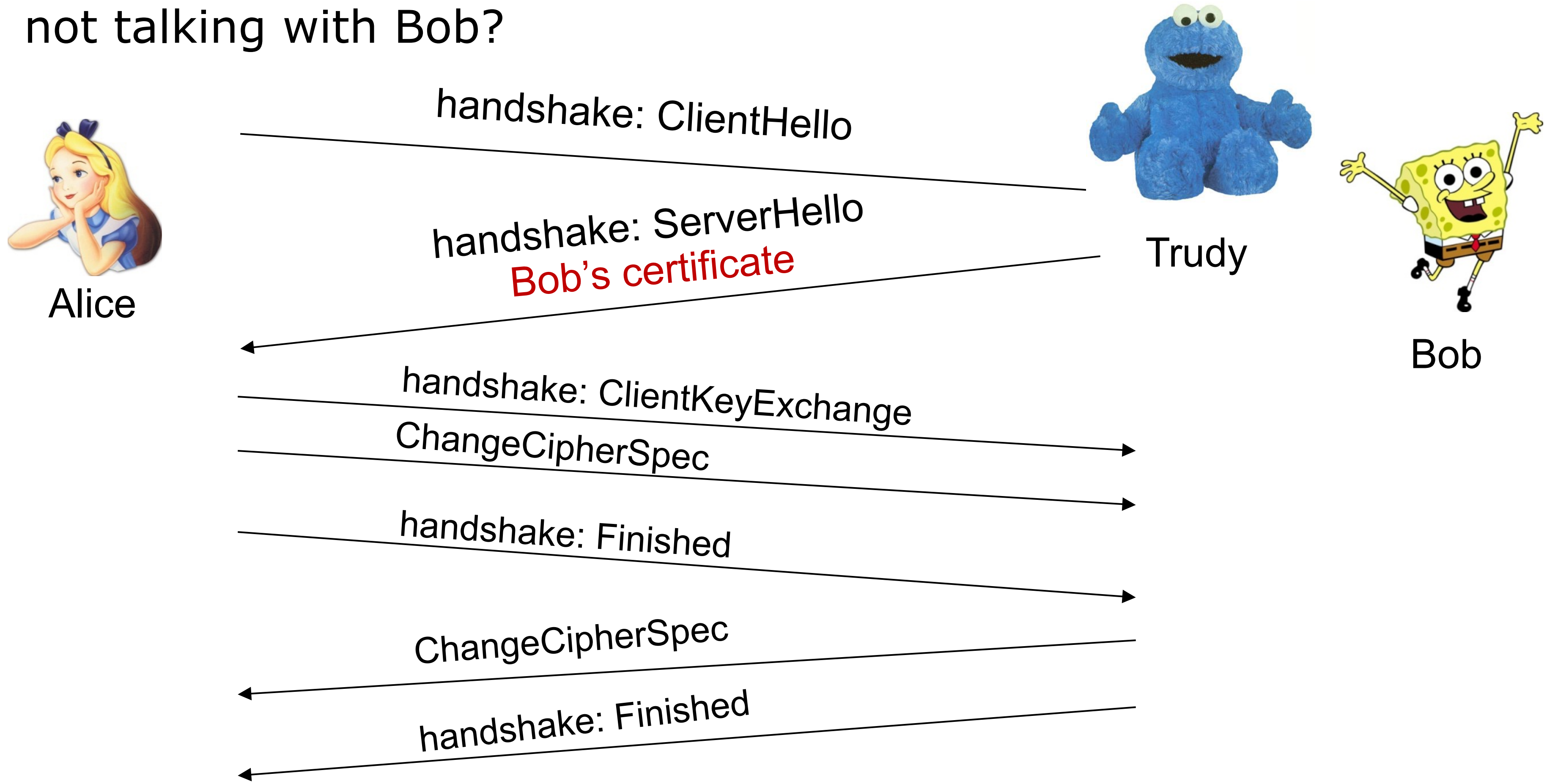
Man-in-the middle could delete the stronger algorithms from list

Last 2 steps prevent this

- Last two messages are encrypted

Short Question

In which step of SSL handshake, can Alice discover that she is not talking with Bob?



SSL: Handshake (contd.)

Why the two random nonces?

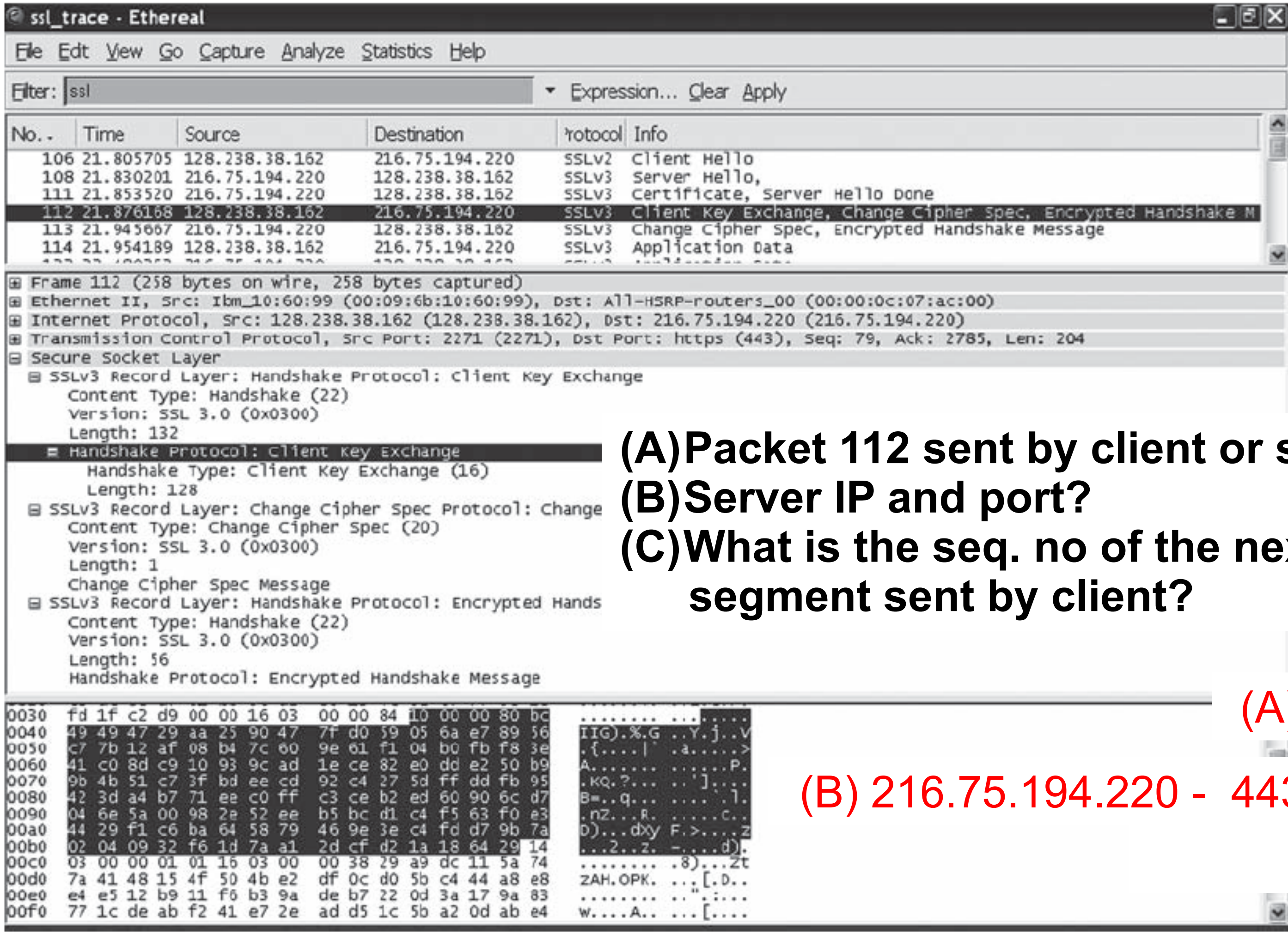
Suppose Trudy sniffs all messages between Alice & Bob.

Next day, Trudy sets up TCP connection with Bob, sends the exact same sequence of records.

- Bob (Amazon) thinks Alice made two separate orders for the same thing.

- Solution: Bob sends different random nonce for each connection. This causes encryption keys to be different on the two days.
- Trudy’s messages will fail Bob’s integrity check.

Questions from an SSL Trace



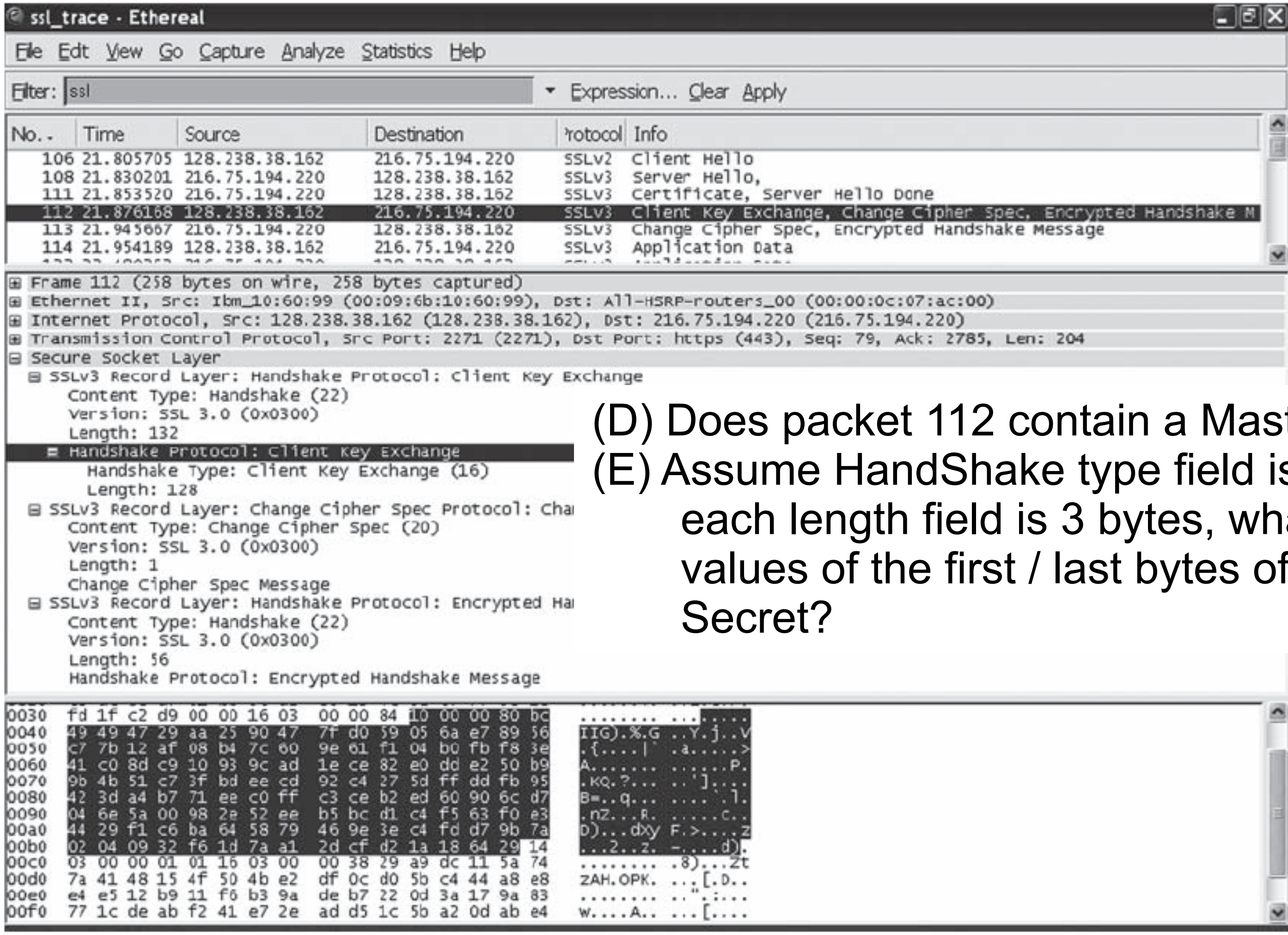
(A)Packet 112 sent by client or server?
(B)Server IP and port?
(C)What is the seq. no of the next TCP segment sent by client?

(A) Client

(B) 216.75.194.220 - 443(https)

Questions from an SSL Trace

SEMO SECURE MOBILE NETWORKING



(D) Does packet 112 contain a Master Secret?

(E) Assume HandShake type field is 1 byte, each length field is 3 bytes, what are the values of the first / last bytes of Master Secret?

Key derivation

Client nonce, server nonce, and pre-master secret input into pseudo random-number generator.

- Produces master secret

Master secret and new nonces inputed into another random-number generator: “key block”

- Because of session resumption: Talk later.

Key block sliced and diced:

- client MAC key
- server MAC key
- client encryption key
- server encryption key
- client initialization vector (IV)
- server initialization vector (IV)

RECALL: Cipher Block Chaining (CBC)

CBC generates its own random numbers

- Have encryption of current block depend on result of previous block
- $c(i) = K_S(m(i) \oplus c(i-1))$
- $m(i) = K_S(c(i)) \oplus c(i-1)$

How do we encrypt first block?

- Initialization vector (IV): random block = c(0)
- IV does not have to be secret

Change IV for each message (or session)

- Guarantees that even if the same message is sent repeatedly, the ciphertext will be completely different each time

SSL Performance

Big-number operations in public-key crypto are CPU intensive

Server handshake

- Typically over half SSL handshake CPU time goes to RSA decryption of the encrypted pre_master_secret

Client handshake

- Public key encryption is less expensive
- Server is handshake bottleneck

Data transfer

- Symmetric encryption
- MAC calculation
- Neither as CPU intensive as public-key decryption

Session resumption

SECURE MOBILE NETWORKING

Full handshake is expensive: CPU time and number of RTTs

If the client and server have already communicated once, they can skip handshake and proceed directly to data transfer

- For a given session, client and server store session_id, master_secret, negotiated ciphers

Client sends session_id in ClientHello

Server then agrees to resume in ServerHello

- New key_block computed from master_secret and client and server random numbers

SSL Handshake Protocol: Abbreviated Handshake

If the server cannot resume / decides not to resume the session it answers with the messages of the full handshake

Client		Server
ClientHello(SessionID)	→	
	←	ServerHello(SessionID) ChangeCipherSpec Finished
ChangeCipherSpec Finished	→	

HTTPS

HTTPS (HTTP over SSL)

- combination of HTTP & SSL/TLS to secure communications between browser & server
  - documented in RFC2818
  - no fundamental change using either SSL or TLS

use https:// URL rather than http://

- and port 443 rather than 80

encrypts

- URL, document contents, form data, cookies, HTTP headers

HTTPS Use

connection initiation

- SSL handshake then HTTP request(s)

connection closure

- have “Connection: close” in HTTP record
- SSL level exchange close_notify alerts
- can then close TCP connection
- must handle TCP close before alert exchange sent or completed

Conclusion

SSL security protocol operates upon and requires a reliable transport service, e.g. TCP

Up to now, security protocols that have been proposed to protect datagram-oriented transport protocols like UDP have not been extremely successful: look if you can find one, where is it used?

Transport layer security protocols offer true end-to-end protection for user data exchanged between application processes

Furthermore, they may interwork with *packet filtering* of today's firewalls

But, protocol header fields of lower layer protocols cannot be protected this way, so they offer no countermeasures to threats to the network infrastructure itself

Acks & Recommended Reading

Selected slides of this chapter courtesy of

- Keith Ross with changes of myself incorporated
- Some other slides courtesy of G. Schäfer (TU Ilmenau) with changes of J. Schmitt (TU Kaiserslautern) and myself incorporated
- Yet some other slides courtesy of R. Perlman, K. Ross, Y. Chen, W. Stallings (L. Brown); changes of myself incorporated

Recommended reading

- [KaPeSp2002] Charlie Kaufman, Radia Perlman, Mike Speciner: Network Security – Private Communication in a Public World, 2nd Edition, Prentice Hall, 2002, ISBN: 978-0-13-046019-6
- [Stallings2014] William Stallings, Network Security Essentials, 4th Edition, Prentice Hall, 2014, ISBN: 978-0-136-10805-4
- [Schäfer2003] G. Schäfer. Netzsicherheit - Algorithmische Grundlagen und Protokolle. dpunkt.verlag, 2003.

Additional References

Readings on SSL

- [BKS98a] D. Bleichenbacher, B. Kaliski, J. Staddon. Recent Results on PKCS #1: RSA Encryption Standard. RSA Laboratories' Bulletin 7, 1998.
- [Cop96a] D. Coppersmith, M. K. Franklin, J. Patarin, M. K. Reiter. Low Exponent RSA with Related Messages. In Advance in Cryptology -- Eurocrypt'96, U. Maurer, Ed., vol. 1070 of Lectures Notes in Computer Science, Springer-Verlag, 1996.
- [FKK96a] A. O. Freier, P. Karlton, P. C. Kocher. The SSL Protocol Version 3.0. Netscape Communications Corporation, 1996.

Copyright Notice

This document has been distributed by the contributing authors as a means to ensure timely dissemination of scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the authors or by other copyright holders, notwithstanding that they have offered their works here electronically.

It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author's copyright. These works may not be reposted without the explicit permission of the copyright holder.

Contact

Prof. Dr.-Ing. Matthias Hollick
Department of Computer Science

SEEMOO
Mornewegstr. 32
64293 Darmstadt/Germany
matthias.hollick@seemoo.tu-darmstadt.de

Phone +49 6151 16-70920
Fax +49 6151 16-70921
www.seemoo.tu-darmstadt.de

SSL Cipher-Suites (1)

No protection (default start value):

- CipherSuite SSL_NULL_WITH_NULL_NULL = { 0x00,0x00 }
- In RFC 5246 is stated "TLS_NULL_WITH_NULL_NULL is specified and is the initial state of a TLS connection during the first handshake on that channel, but MUST NOT be negotiated, as it provides no more protection than an unsecured connection."

Server provides an RSA key suitable for encryption:

- SSL_RSA_WITH_NULL_MD5 = { 0x00,0x01 }
- SSL_RSA_WITH_NULL_SHA = { 0x00,0x02 }
- SSL_RSA_EXPORT_WITH_RC4_40_MD5 = { 0x00,0x03 }
- SSL_RSA_WITH_RC4_128_MD5 = { 0x00,0x04 }
- SSL_RSA_WITH_RC4_128_SHA = { 0x00,0x05 }
- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5 = { 0x00,0x06 }
- SSL_RSA_WITH_IDEA_CBC_SHA = { 0x00,0x07 }
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA = { 0x00,0x08 }
- SSL_RSA_WITH_DES_CBC_SHA = { 0x00,0x09 }
- SSL_RSA_WITH_3DES_EDE_CBC_SHA = { 0x00,0x0A }

SSL Cipher-Suites (2)

Cipher-Suites with an authenticated DH-Key-Exchange

- SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA = { 0x00,0x0B }
- SSL_DH_DSS_WITH_DES_CBC_SHA = { 0x00,0x0C }
- SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA = { 0x00,0x0D }
- SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA = { 0x00,0x0E }
- SSL_DH_RSA_WITH_DES_CBC_SHA = { 0x00,0x0F }
- SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA = { 0x00,0x10 }
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA = { 0x00,0x11 }
- SSL_DHE_DSS_WITH_DES_CBC_SHA = { 0x00,0x12 }
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA = { 0x00,0x13 }
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA = { 0x00,0x14 }
- SSL_DHE_RSA_WITH_DES_CBC_SHA = { 0x00,0x15 }
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA = { 0x00,0x16 }
  - DH stands for suites in which the public DH values are contained in a certificate signed by a CA
  - DHE for suites in which they are signed with a public key which is certified by a CA

SSL Cipher-Suites (3)

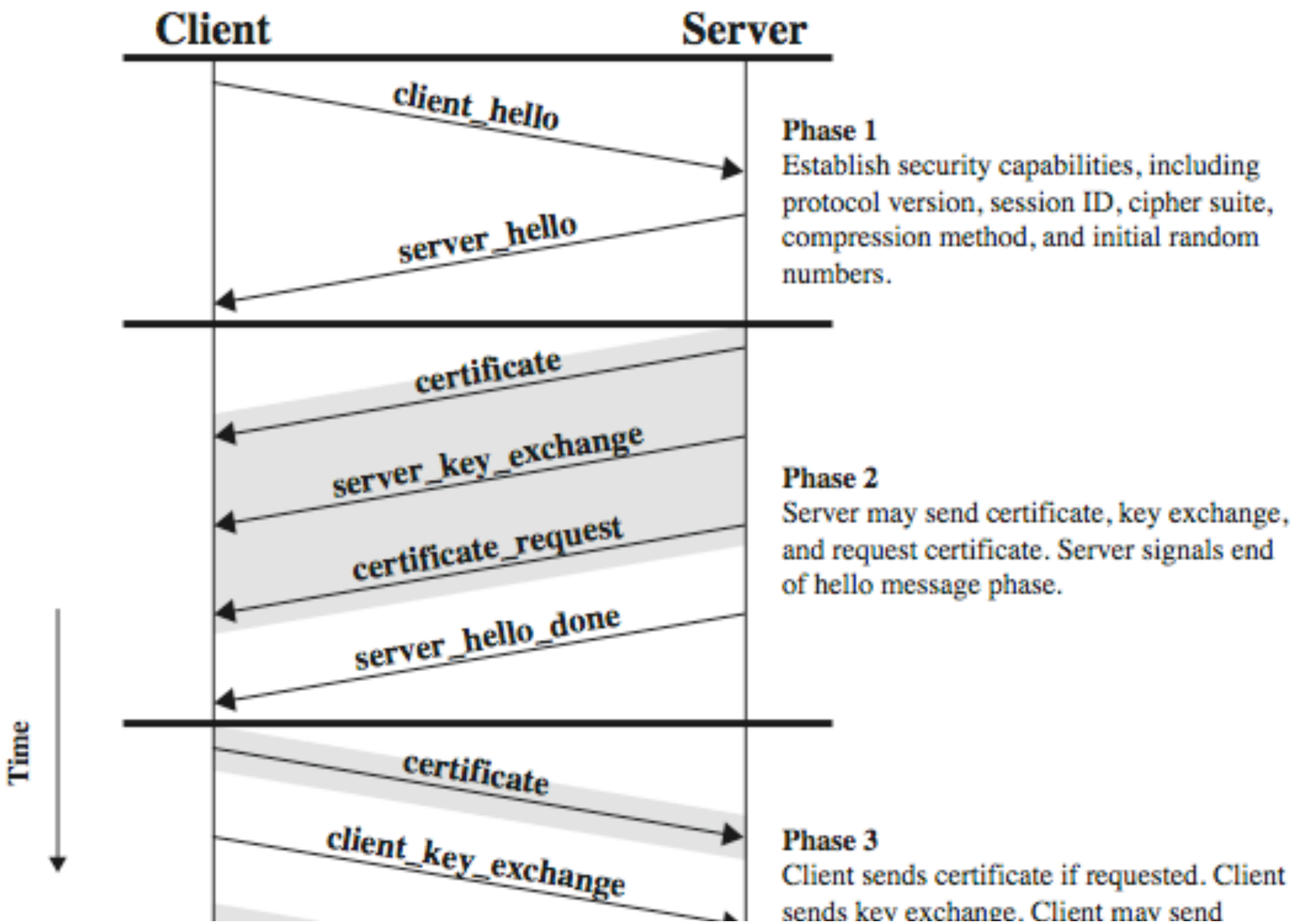
The use of the following cipher-suites without any entity authentication is strongly discouraged, as they are vulnerable to man-in-the-middle attacks:

- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5 = { 0x00,0x17 }
- SSL_DH_anon_WITH_RC4_128_MD5 = { 0x00,0x18 }
- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA = { 0x00,0x19 }
- SSL_DH_anon_WITH_DES_CBC_SHA = { 0x00,0x1A }
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA = { 0x00,0x1B }

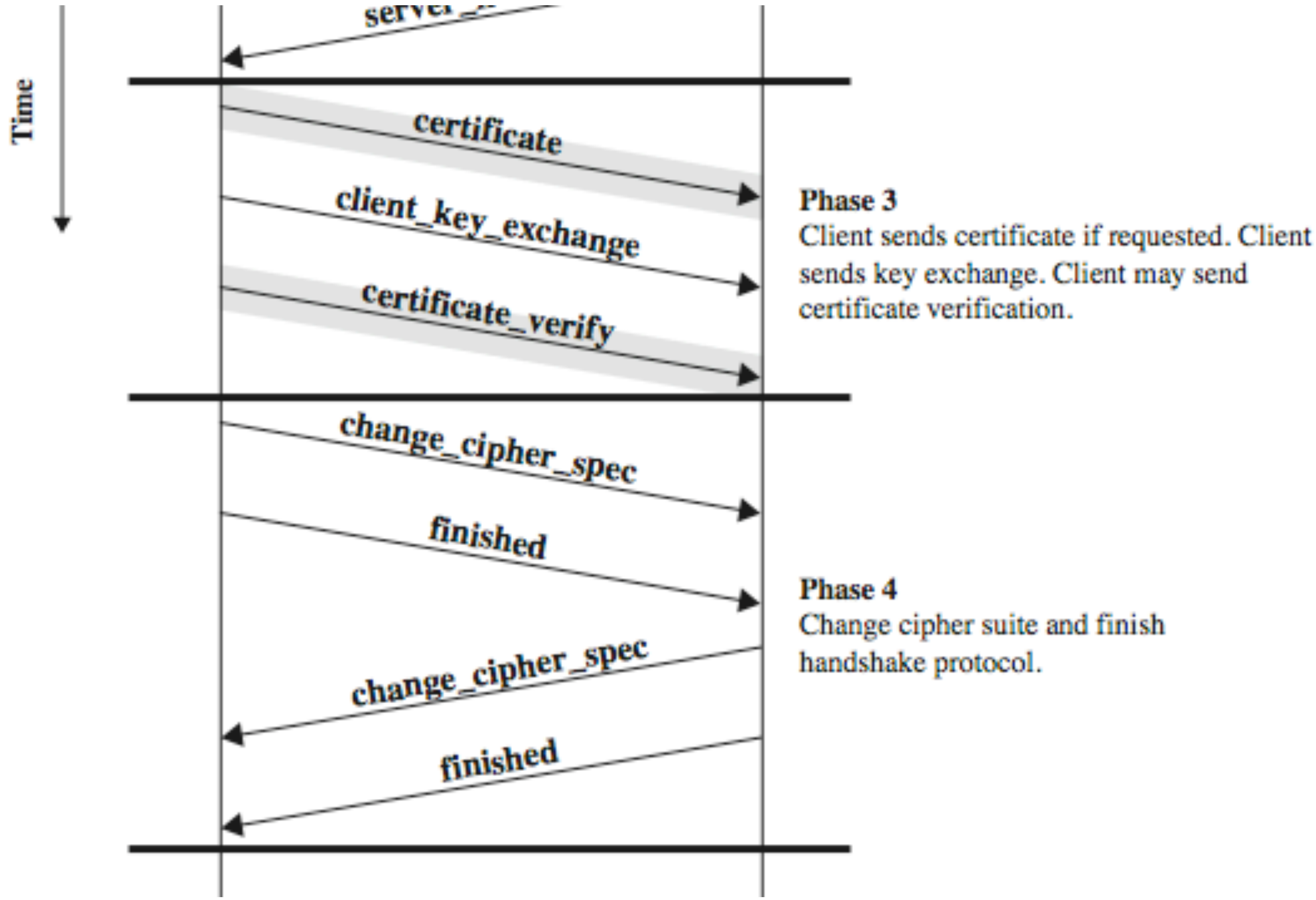
Check out RFC5246 to find even more in the latest TLS specs such as

- TLS_DH_anon_WITH_AES_256_CBC_SHA256 = { 0x00,0x6D };

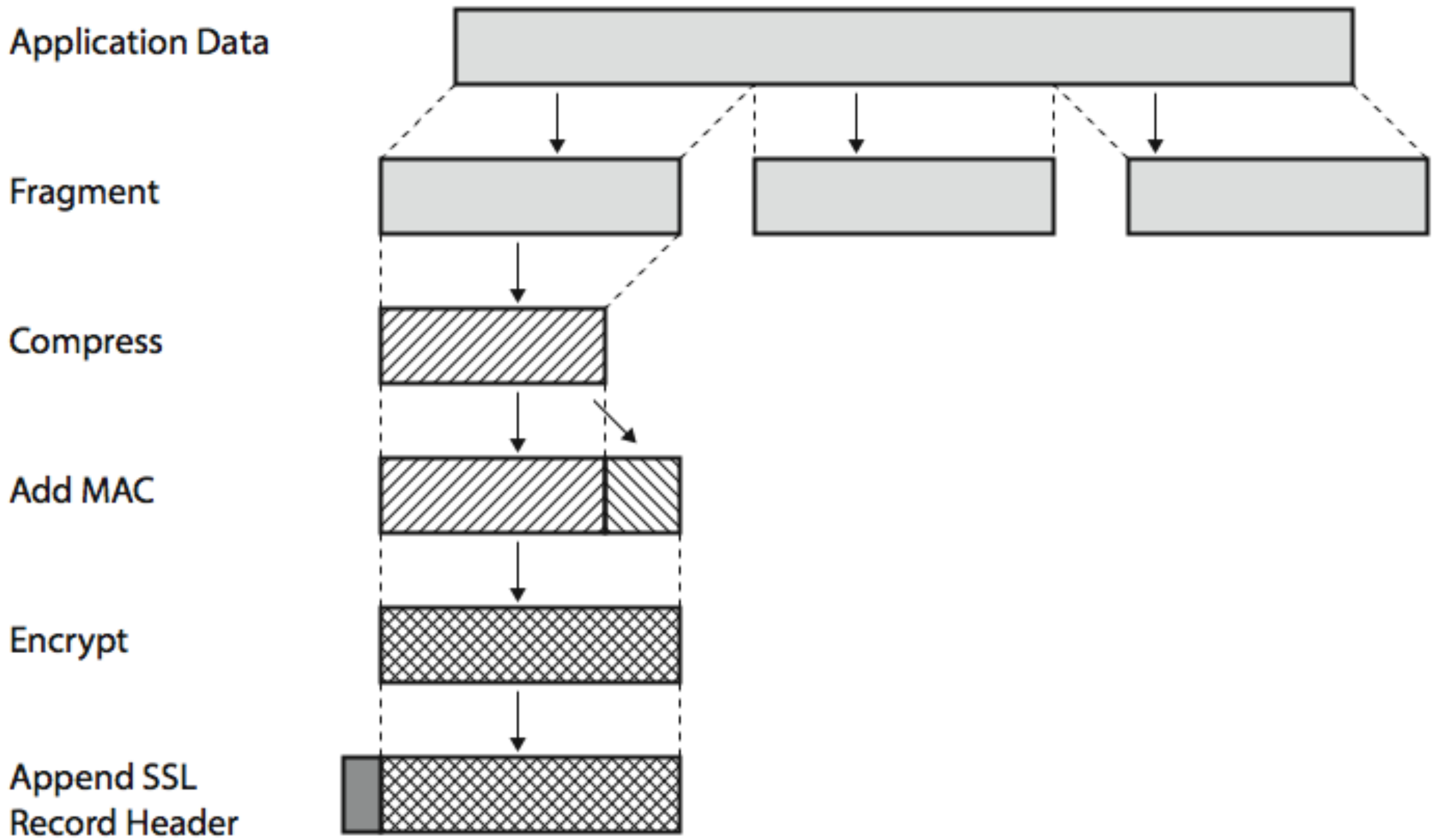
Yet another Visualization of SSL Handshake Protocol



Yet another Visualization of SSL Handshake Protocol



SSL Record Protocol Operation



SSL Change Cipher Spec Protocol

one of 3 SSL specific protocols which use the SSL Record protocol

- a single message
- causes pending state to become current
- hence updating the cipher suite in use

(a) Change Cipher Spec Protocol

SSL Alert Protocol

conveys SSL-related alerts to peer entity

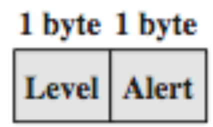
severity

- warning or fatal

specific alert

- fatal: unexpected message, bad record mac, decompression failure, handshake failure, illegal parameter
- warning: close notify, no certificate, bad certificate, unsupported certificate, certificate revoked, certificate expired, certificate unknown

compressed & encrypted like all SSL data



(b) Alert Protocol

SSL Handshake Protocol

allows server & client to:

- authenticate each other
- to negotiate encryption & MAC algorithms
- to negotiate cryptographic keys to be used

comprises a series of messages in phases

- Establish Security Capabilities
- Server Authentication and Key Exchange
- Client Authentication and Key Exchange
- Finish

1 byte	3 bytes	≥ 0 bytes
Type	Length	Content

(c) Handshake Protocol