Security Engineering
Part III – Network Security

Security Protocols (I): SSL/TLS

Juan E. Tapiador
jestevez@inf.uc3m.es
Department of Computer Science, UC3M

Preliminaries

SSL: Secure Socket Layer (Netscape)

- v1 (1990-1994). Internal use. Never released
- v2 (1994). Usable but lots of security vulnerabilities
- v3 (1996). Stable. Basis for TLS 1.0

TLS: Transport Layer Security

- RFC 2246
- Free open-source implementation at http://www.openssl.org
- TLS 1.0 aprox. = SSL 3.0 with some changes

SSL/TLS gives transport-layer security

- TCP (reliable end-to-end transport required)
- Applications must undergo slightly modifications

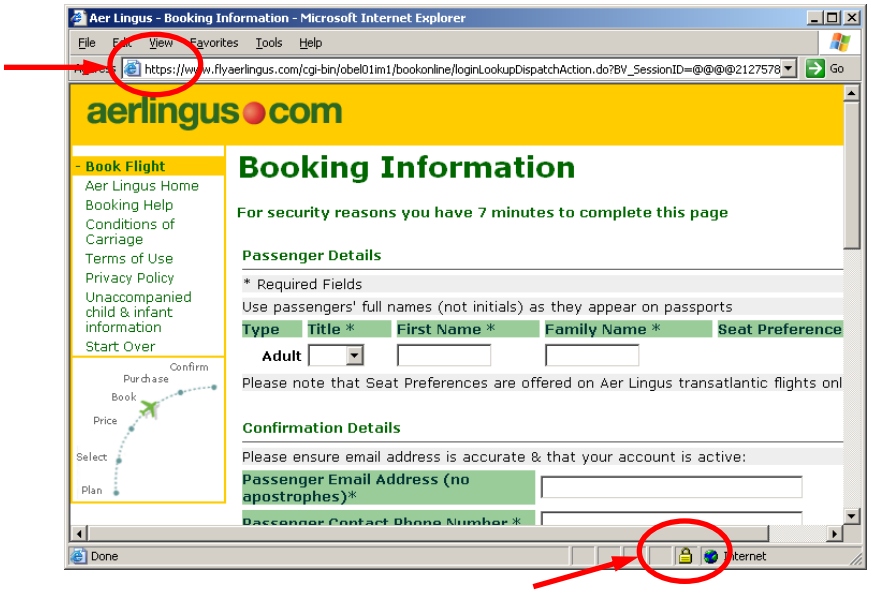
Preliminaries

SSL usage:

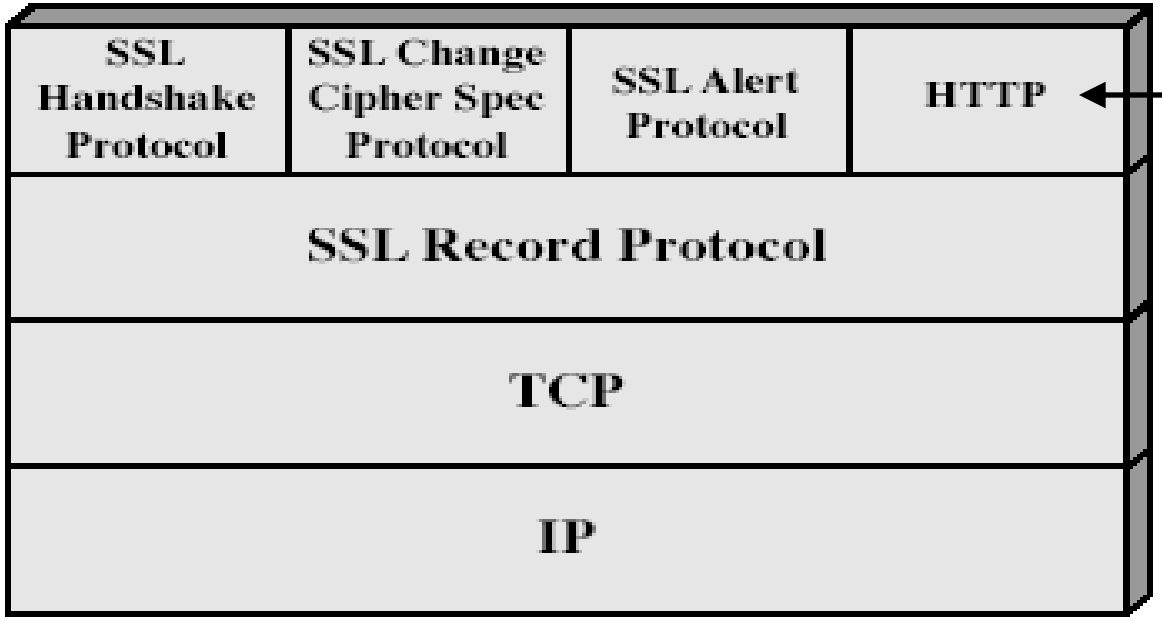
- Encryption of messages exchanged between client and server
  - E.g., credit card numbers, e-mails, online banking, ...)
- Authentication of communication parties:
  - Ensure server authenticity
  - Ensure client authenticity (optional)

SSL in HTTP transactions:

- https://·····
- Lock displayed in browser
- Maybe a warning



SSL – Architecture



SSL Record Protocol: authentication and encryption.

Upper layers: session establishment, parameter negotiation, etc.

SSL – Session vs Connection

SSL session

- Association between a client and a server
  - ($host_S$, $port_S$, $host_C$, $port_C$)
- Created by the Handshake Protocol
- Defines a number of shared cryptographic parameters
  - Algorithms, master secret (MS), certificates, key lengths, ...
- Can be shared by multiple SSL connections
- *Idea: avoid repeated used of Handshake Protocol (costly)*

SSL connection

- Transient communication link
- Associated with an SSL session
- Stated defined by: nonces, secret keys used by MAC and encryption algorithms, IVs, sequence numbers.
- Keys derived from MS created during Handshake.

SSL Handshake Protocol

SSL uses symmetric cryptography for:

- MACs and encryption (Record Protocol)
- Different keys for each communication party!

In addition to key establishment, the Handshake Protocol has among its goals:

- Client and server authentication
  - Server almost always authenticated
  - Client rarely authentication, though possible
  - Useful model for most e-commerce applications
- Secure negotiation of the ciphersuite (algorithms + parameters)
  - Encryption
  - Hash function
  - Authentication method
  - Key establishment

SSL Handshake Protocol

Key establishment

- Various mechanisms supported
  - *RSA-based*: C chooses PMS and sends it to S using S's public key
  - *Diffie-Hellman*

Entity authentication

- Again various mechanisms supported:
  - *RSA-based*: ability to correctly decrypt PMS and generate a valid MAC (using keys derived from PMS). This implicitly authenticates S

SSL Handshake Protocol

Key derivation

- Keys used by Record Protocol for encryption and MAC are derived from PMS:
  - MS obtained from (PMS, client nonce, server nonce) through a hash function
  - Key_block obtained from (MS, client nonce, server nonce) through repeated application of hash function
  - Keys obtained from Key_block

SSL Handshake Protocol

Protocol messages:

Usual SSL configuration:

- No client authentication
- Cliente sends PMS using S's RSA public key, obtained from its certificate
- Server authenticated if capable of decrypting PMS and construct a valid *finish* message

M1: C → S: *ClientHello*

- Cliente starts connection
- Sends version number
- Sends *ClientNonce* (28 random bytes+ 4-byte timestamp)
- Sends ciphersuite (key exchange, authentication methods, encryption and MAC algorithms, hash funcitons)

SSL Handshake Protocol

M2: S → C: *ServerHello, ServerCertChain, ServerHelloDone*

- Server sends version number
- Sends *ServerNonce* and *SessionID*
- Chooses ciphersuite from C's list
- *ServerCertChain*: needed by client to verify S's public key through TTP
- (optional) *CertRequest*: if C to be authenticated
- Finally, *ServerHelloDone*

M3: C → S: *ClientKeyExchange, ChangeCipherSpec, ClientFinished*

- *ClientKeyExchange*: contains encrypted PMS
- *ChangeCipherSpec*: from now on, everything encrypted+authenticated
- (optional) *ClientCertificate, ClientCertificateVerify*
- Finally, *ClientFinished*
  - Contains MAC of all messages (both directions) exchanged so far.

SSL Handshake Protocol

M4: S → C: *ChangeCipherSpec, ServerFinished*

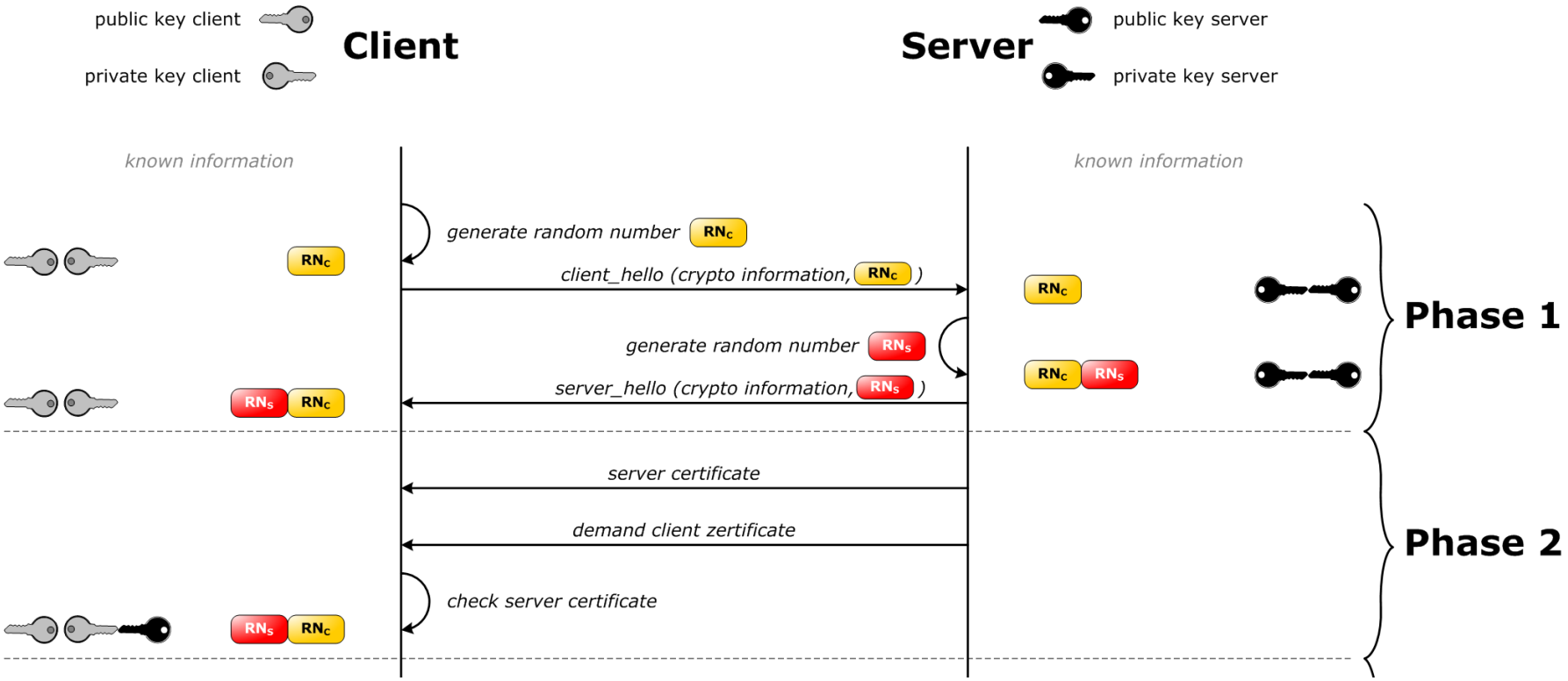
- Server starts encryption+authentication
- *ServerFinished* contains:
  - MAC of all messages exchanged (both directions) so far
  - Correct key_block needed!
  - Only computable if S has obtained MS and has decrypted M3

There's a possibility to renegotiate the ciphersuite:

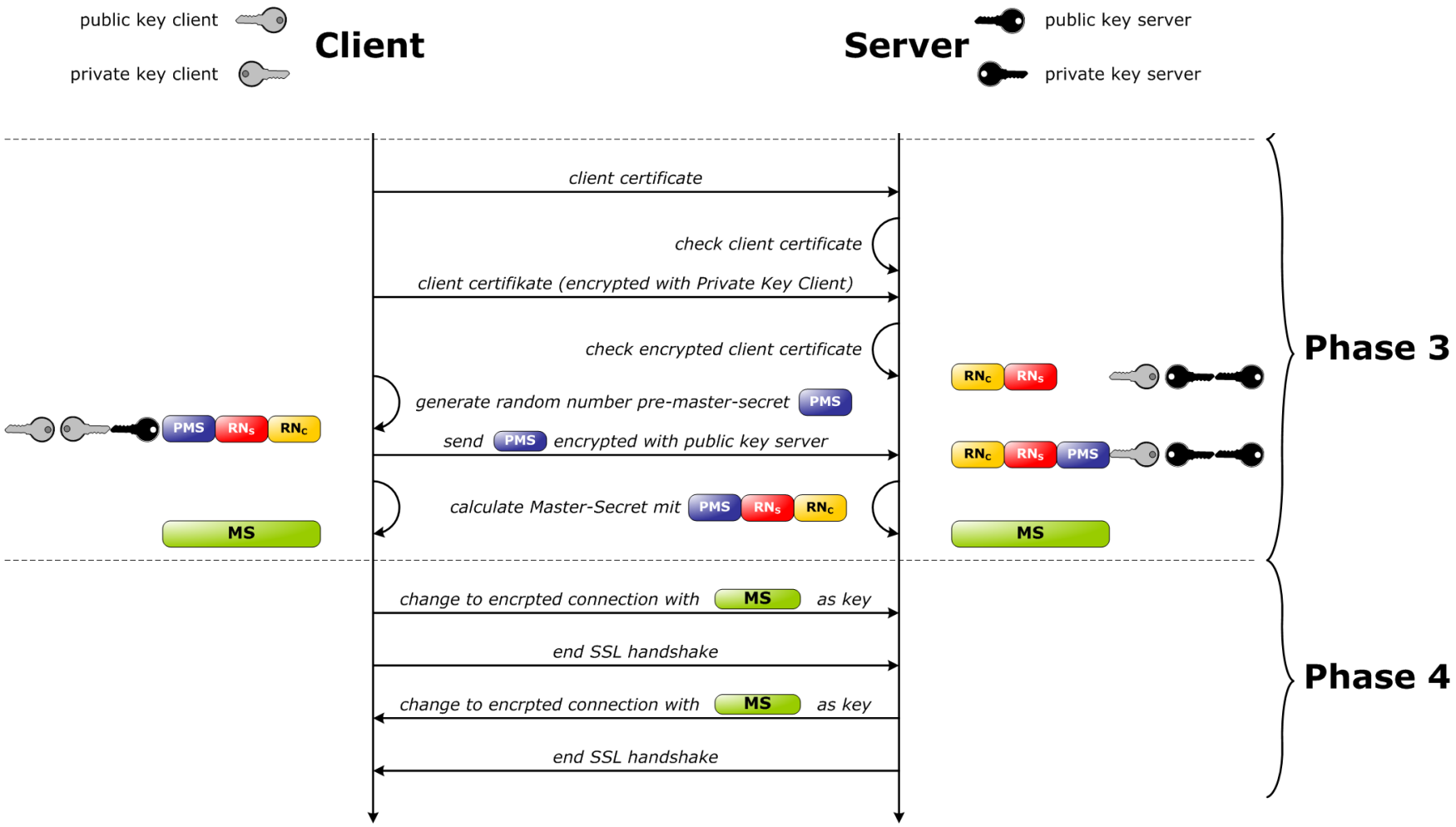
- Useful to reuse MS over multiple connections…
- … but keys are recomputed using new nonces
- Can be done during one session
- Protected by Record Protocol

SSL Handshake Protocol

Source: Wikipedia



SSL Handshake Protocol



Source: Wikipedia

SSL Alert & Change Ciphersuite

Alert Protocol

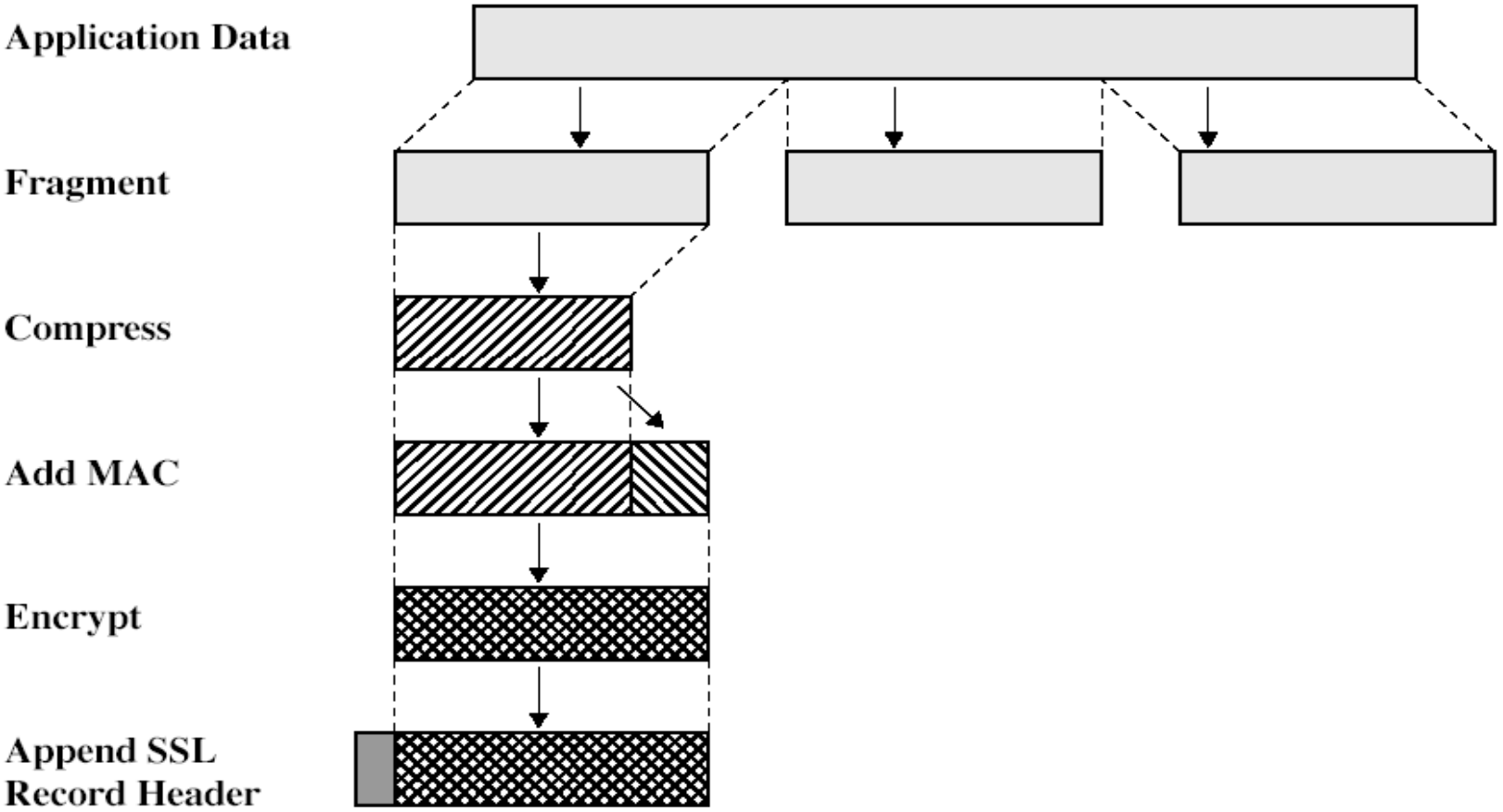
- Manages SSL session and error/warning messges:
- *Unexpected message*
  - *Bad record MAC*
  - *Decompression failure*
  - *Bad certificate*
  - *Certificate revoked*
  - *Certificate expired*
  - Etc...

Change Ciphersuite Protocol

- 1 message only
- Used to announce that one party will immediately change to the recently negotiated ciphersuite

Both executed on top of Record Protocol

SSL Record Protocol



Source: Stallings

SSL Record Protocol

Confidentiality

- Symmetric encryption
- IDEA, RC2-40, DES-40, DES, 3DES, Fortezza, RC4-40, ...
- Message is optionally compressed before encryption, e.g. using Lempel-Ziv (ZIP)

Integrity & message authentication

- MAC (with a shared key)

SSL vs TLS

- Standard RFC 2246, similar to SSL 3.0
- TLS version numbers:
  - SSL 3.1 (TLS 1.0)
  - SSL 3.2 (TLS 1.1)
  - SSL 3.3 (TLS 1.2)
- Uses HMAC as MAC algorithm
- Different method for deriving *master_secret* and *key_block*
  - PRF based on HMAC with MD5 or SHA-1
- More warning and error messages
- More client certificates supported
- Variable-length padding
  - Used to hide the length of short messages
  - Protects against traffic analysis attacks
- More crypto algorithms supported
- Etc...