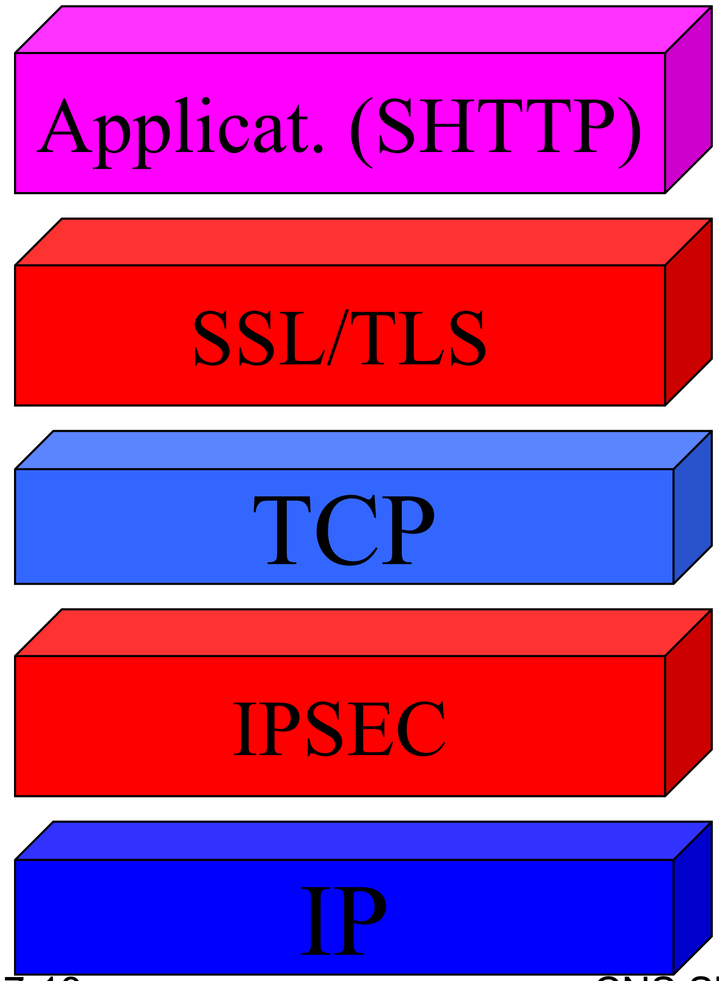
Computer and Network Security

SSL/TSL

Security architecture and protocol stack



Secure applications: PGP, SHTTP, SFTP,…

or

Security down in the protocol stack

-SSL between TCP and applic. layer

- IPSEC between TCP and IP

SSL/TLS intro

- Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that provide security for communications over networks
- TLS and SSL encrypt the segments of network connections at the Transport Layer end-to-end

SSL/TLS intro, continued

- Several versions of the protocols are in widespread use in applications like web browsing, electronic mail, Internet faxing, instant messaging and VoIP
- TLS is an IETF standards track protocol, last updated in RFC 5246 (2008) and in RFC 6176 ("Prohibiting SSL V. 2.0", 2011)
- TLS derives from the earlier SSL specifications developed by Netscape Corporation (Taher El-Gamal)

SSL/TLS intro, continued 2

- SSL/TLS allow client/server applications to communicate across a network in a way designed to prevent eavesdropping, tampering and message forgery
- provide endpoint authentication and communications confidentiality over the Internet using cryptography
  - RSA security with 1024 and 2048 bit strengths
- current version TLS 1.2 (RFC 5246, many updates)
  - SSL 1, 2, 3 broken
  - TLS 1.0 having some weakness;
  - TLS 1.1 & 1.2 good;
  - TLS 1.3 available as a draft (July 2017)

main threats addressed

- eavesdropping = the act of secretly listening to private conversation
- tampering = the act of altering something secretly or improperly
- message forgery = sending of a message to deceive the recipient as to whom the real sender is

SSL/TLS intro, continued 3

- In typical end-user/browser usage, TLS authentication is unilateral: only server is authenticated, but not vice
- TLS also supports mutual authentication
  - provided that partners diligently scrutinize identity information

SSL/TLS intro, continued 4

- Mutual authentication requires that the TLS client-side also holds a certificate (which is not usually the case in the end-user/browser scenario)
  - Unless TLS-PSK, the Secure Remote Password (SRP) protocol, or some other protocol is used that can provide strong mutual authentication in the absence of certificates

SSL/TLS intro, continued 5

TLS involves three basic phases:

1. Peer negotiation for algorithm support
2. Key exchange and authentication
3. Symmetric cipher encryption and message authentication

SSL/TLS intro, continued 6

- During first phase, client and server negotiate cipher suites, which determine ciphers to be used, key exchange and authentication algorithms, as well as message authentication codes (MACs)
  - key exchange and authentication algorithms are typically public key algorithms, or, as in TLS-PSK, preshared keys (PSKs) could be used
  - message authentication codes are made up from cryptographic hash functions using the HMAC construction for TLS, and a non-standard pseudorandom function for SSL.

SSL/TLS: typical algorithms

- For key exchange: RSA, Diffie-Hellman, ECDH (Elliptic Curve Diffie–Hellman), SRP (Secure Remote Password protocol), PSK
- For authentication: RSA, DSA, ECDSA (Elliptic Curve Digital Signature Algorithm)
- Symmetric ciphers: RC4, Triple DES, AES, IDEA, DES, or Camellia. In older versions of SSL, RC2 was also used.
- For cryptographic hash function: HMAC-MD5 or HMAC-SHA are used for TLS, MD5 and SHA for SSL, while older versions of SSL also used MD2 and MD4.

SLL/TLS and digital certificates

- The key information and certificates necessary for TLS are handled in the form of X.509 certificates, which define required fields and data formats.

how SSL/TLS works 1/5

- Client and server negotiate a stateful connection by using a handshaking procedure. During handshake, client and server agree on various parameters used to establish connection's security
- Handshake begins when client connects to TLS-enabled server requesting a secure connection and presents a list of supported ciphers and hash functions
- From this list, server picks the strongest cipher and hash function that it also supports and notifies client of the decision

how SSL/TLS works 2/5

- Server sends back its identification in the form of a digital certificate X.509
- Client may contact the CA and confirm that the certificate is authentic and not revoked before proceeding
  - modern browsers support Extended Validation certificates and can use the Online Certificate Status Protocol (OCSP)

how SSL/TLS works 3/5

- For generating session keys used for secure connection, client encrypts a random number (RN) with server's public key (PbK), and sends result to server.
  - Only server is able to decrypt it (with its private key (PvK)): this is the one fact that makes the keys hidden from third parties, since only the server and the client have access to this data.

how SSL/TLS works 4/5

- Client knows PbK and RN, and server knows PvK and (after decryption of the client's message) RN. A third party may only know PbK, unless PvK has been compromised.
- From the random number, both parties generate key material for encryption and decryption.

how SSL/TLS works 5/5

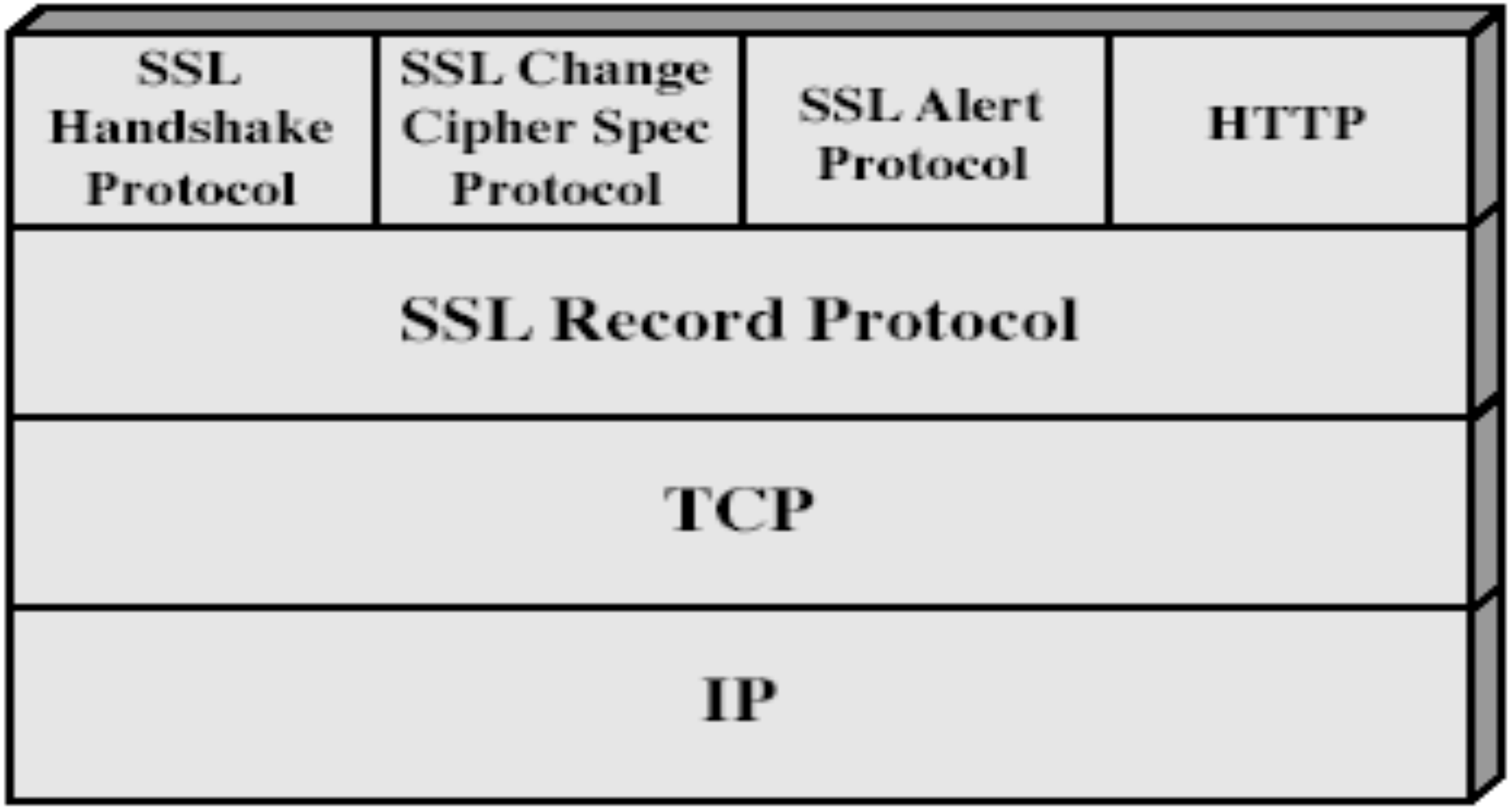
- This concludes the handshake and begins the secured connection, which is encrypted and decrypted with the key material until the connection closes.
- *If any one of the above steps fails, the TLS handshake fails, and the connection is not created.*

SSL (Secure Socket Layer)

- transport layer security service
- uses TCP to provide a reliable end-to-end service
  - originally developed by Netscape
  - version 3 designed with public input
  - subsequently became Internet standard known as TLS (Transport Layer Security)

- SSL has two layers of protocols

SSL Architecture



SSL Architecture

- SSL session
  - an association between client & server
  - created by the Handshake Protocol
  - defines a set of cryptographic parameters
  - maybe shared by multiple SSL connections (re-negotiating can be onerous)
  - stateful
- SSL connection
  - a transient, peer-to-peer, communications link
  - associated with 1 SSL session
  - stateful

sessions and connections

- between any pair of parties there may be multiple secure connections
  - there may also be multiple simultaneous sessions between parties, but this feature is not used in practice
- several states associated with each session
  - once a session is established, there is a current operating state for both read and write (i.e., receive and send)
  - during Handshake Protocol, pending read and write states are created
  - after conclusion of Handshake Protocol, the pending states become the current states

parameters defining session state

- Session identifier

  arbitrary byte sequence chosen by the server to identify an active or resumable session state
- Peer certificate

  X509.v3 certificate of the peer. This element of the state may be null
- Compression method

  The algorithm used to compress data prior to encryption.

parameters defining session state

- Cipher spec

  Specifies the bulk data encryption algorithm (such as null, DES, etc.) and hash algorithm (such as MD5 or SHA-l) used for MAC calculation. It also defines cryptographic attributes such as the hash_size.
- Master secret

  48-byte secret shared between the client and server.
- Is resumable

  flag indicating whether the session can be used to initiate new connections.

parameters defining connection state

- Server and client random

  Byte sequences that are chosen by the server and client for each connection.
- Server write MAC secret

  The secret key used in MAC operations on data sent by the server
- Client write MAC secret

  The secret key used in MAC operations on data sent by the client.
- Server write key

  The conventional encryption key for data encrypted by the server and decrypted by the client

parameters defining connection state

- Client write key

  The conventional encryption key for data encrypted by the client and decrypted by the server.

- Initialization vectors

  When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the SSL Handshake Protocol. Thereafter the final ciphertext block from each record is preserved for use as the IV with the following record

parameters defining connection state

- Sequence numbers

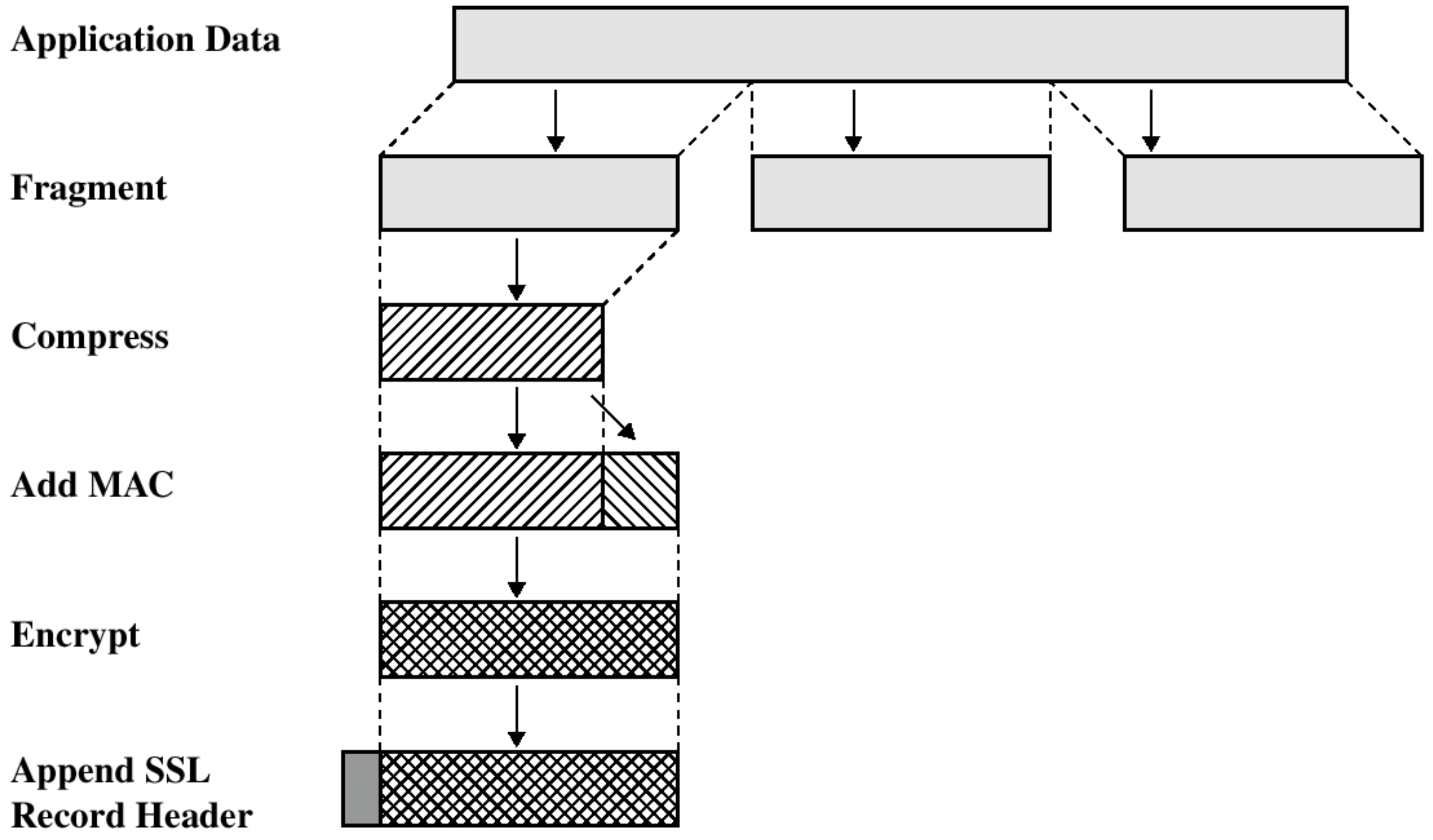
  Each party maintains separate sequence numbers for transmitted and received messages for each connection. When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero. Sequence numbers may not exceed $2^{64} - 1$.

SSL Record Protocol

two main services

- **confidentiality**
  - using symmetric encryption with a shared secret key defined by Handshake Protocol
  - IDEA, RC2-40, DES-40, DES, 3DES, Fortezza, RC4-40, RC4-128
  - message is compressed before encryption
- **message integrity**
  - using a MAC with shared secret key
  - similar to HMAC but with different padding

SSL - Record Protocol



Authentication: MAC

Similar to HMAC (uses concatenation instead of EXOR)

```
Hash(MAC_secret_key || pad2
        ||hash(MAC_secret_key || pad1 || seqNum ||
        SSLcompressed.type ||
        SSLcompressed.length ||
        SSLcompressed.fragment))
```

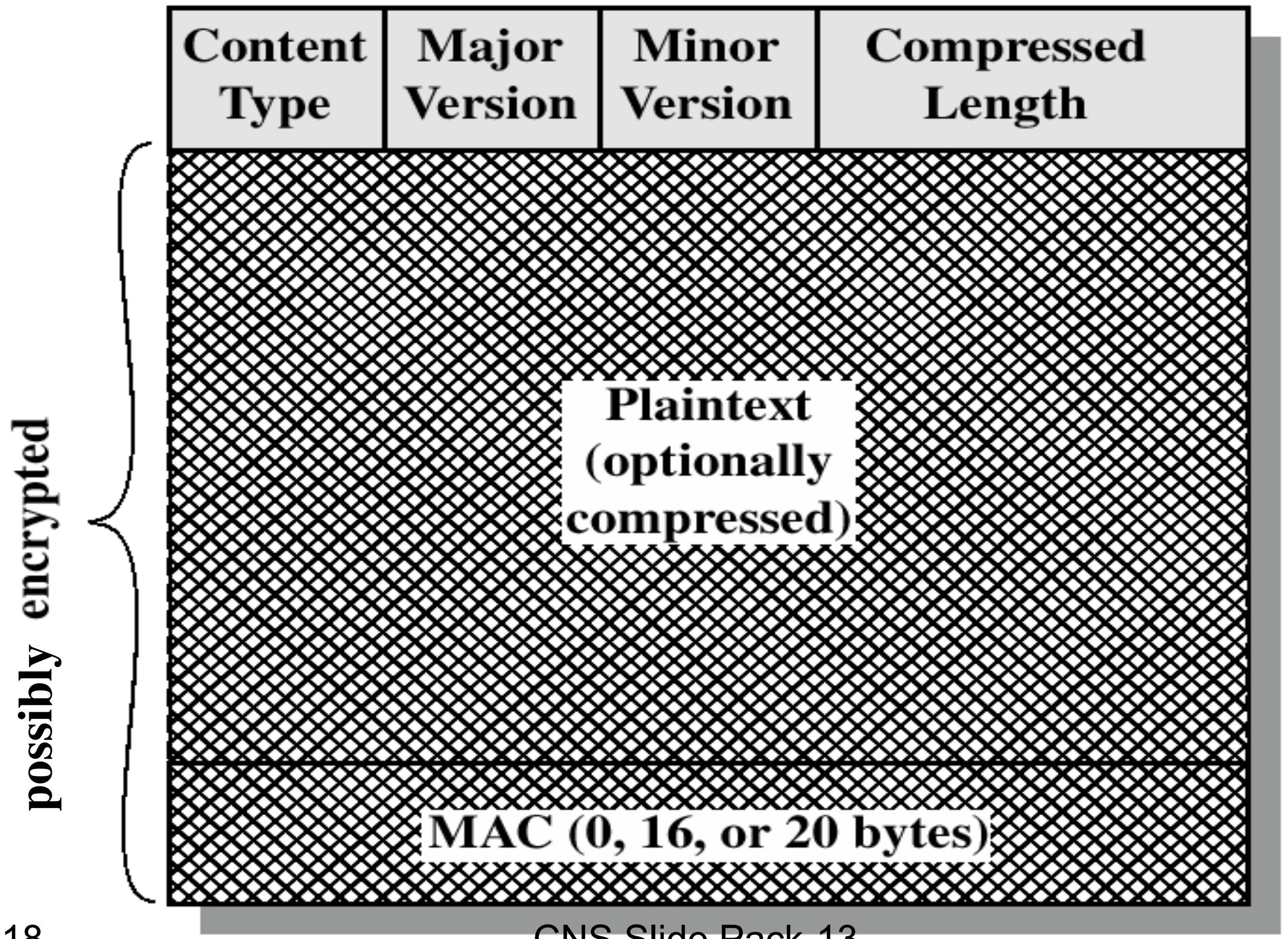
- pad1=0x36 repeated 48 times (MD5); 40 times (SHA-1)
- pad2=0x5C repeated ...
- SSLcompressed.type = high level protocol used to process segment

encoding methods

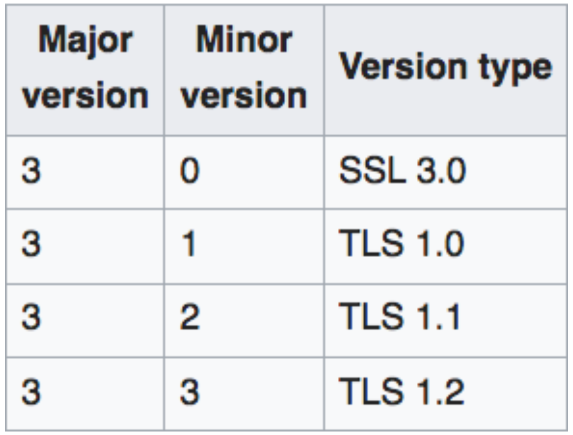
- segment into blocks of 2^{14} = 16384 bytes
- compression (optional):
  - must be no lossy and must guarantee to reduce pack size
  - default in SSLv3 : no compression
- MAC computation (see previous slide)
  - on compressed data
- several (symmetric) encryption methods:
  - block ciphers: IDEA (128) RC2-40, DES-40, DES (56), 3DES (168),
  - Stream Cipher: RC4-40, RC4-128
  - Smart card: Fortezza

SSL - record

fields of the header



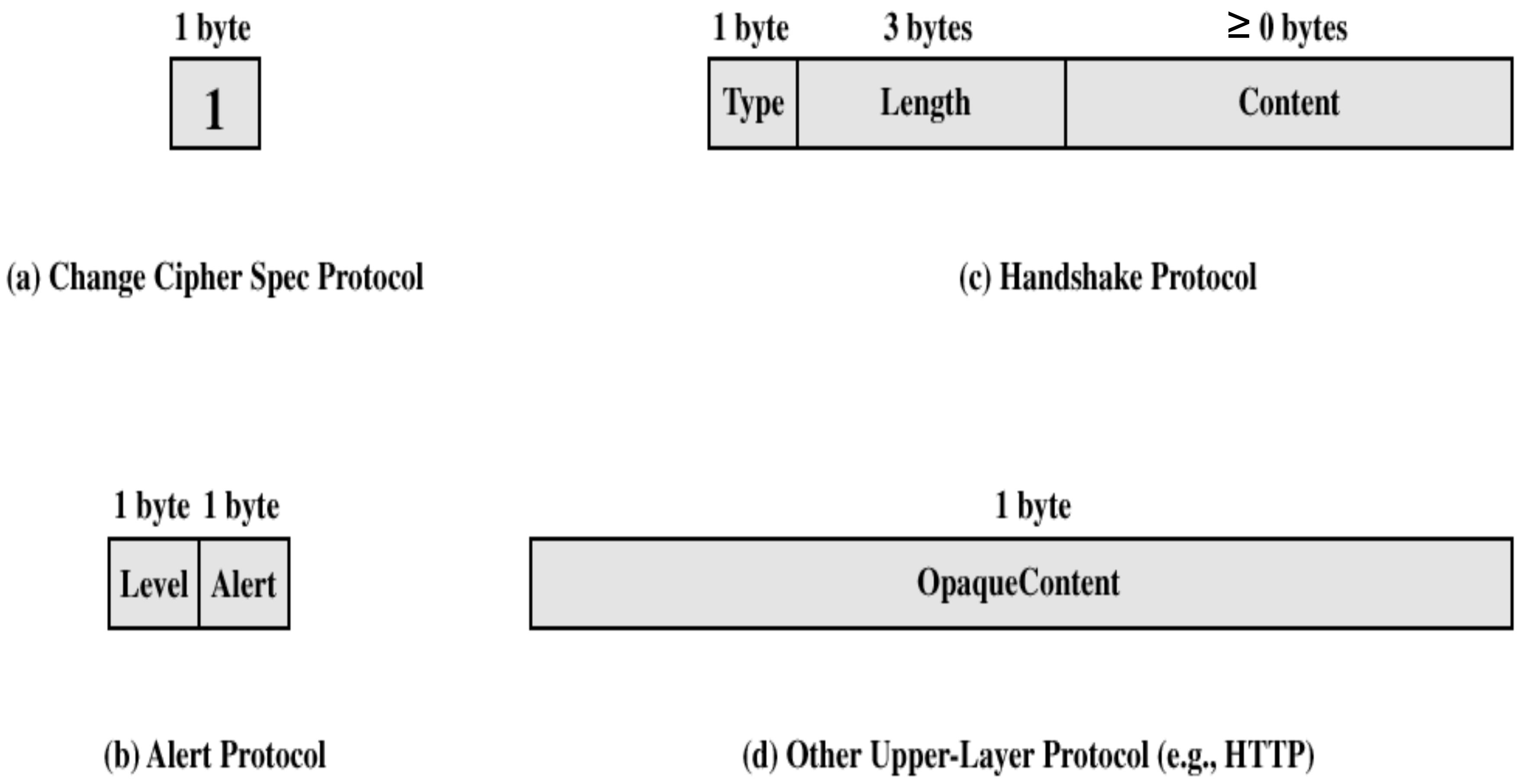
fields


Versions

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

- Content Type (8 bits)
  - The higher layer protocol used to process the enclosed fragment (change_cipher_spec, alert, handshake, and application_data. The first three are the SSL-specific protocols; no distinction is made among the various applications (e.g., HTTP) that might use SSL)
- Major Version (8 bits)
  - Indicates major version of SSL in use. For SSLv3 and TLS1.2, the value is 3
- Minor Version (8 bits)
  - Indicates minor version in use. For SSLv3, the value is 0; for TLS1.2 it is 3
- Compressed Length (16 bits)
  - The length in bytes of the plaintext fragment (or compressed fragment if compression is used).

SSL - Payload


1 byte
1
(a) Change Cipher Spec Protocol
1 byte
3 bytes
≥ 0 bytes
Type
Length
Content
(c) Handshake Protocol
1 byte
1 byte
Level
Alert
(b) Alert Protocol
1 byte
OpaqueContent
(d) Other Upper-Layer Protocol (e.g., HTTP)

SSL Change Cipher Spec Protocol

- one of 3 SSL specific protocols which use the SSL Record protocol
- a single message
- to cause the pending state to be copied into the current state, which updates the cipher suite to be used on this connection
- usually sent just after handshaking

SSL Alert Protocol

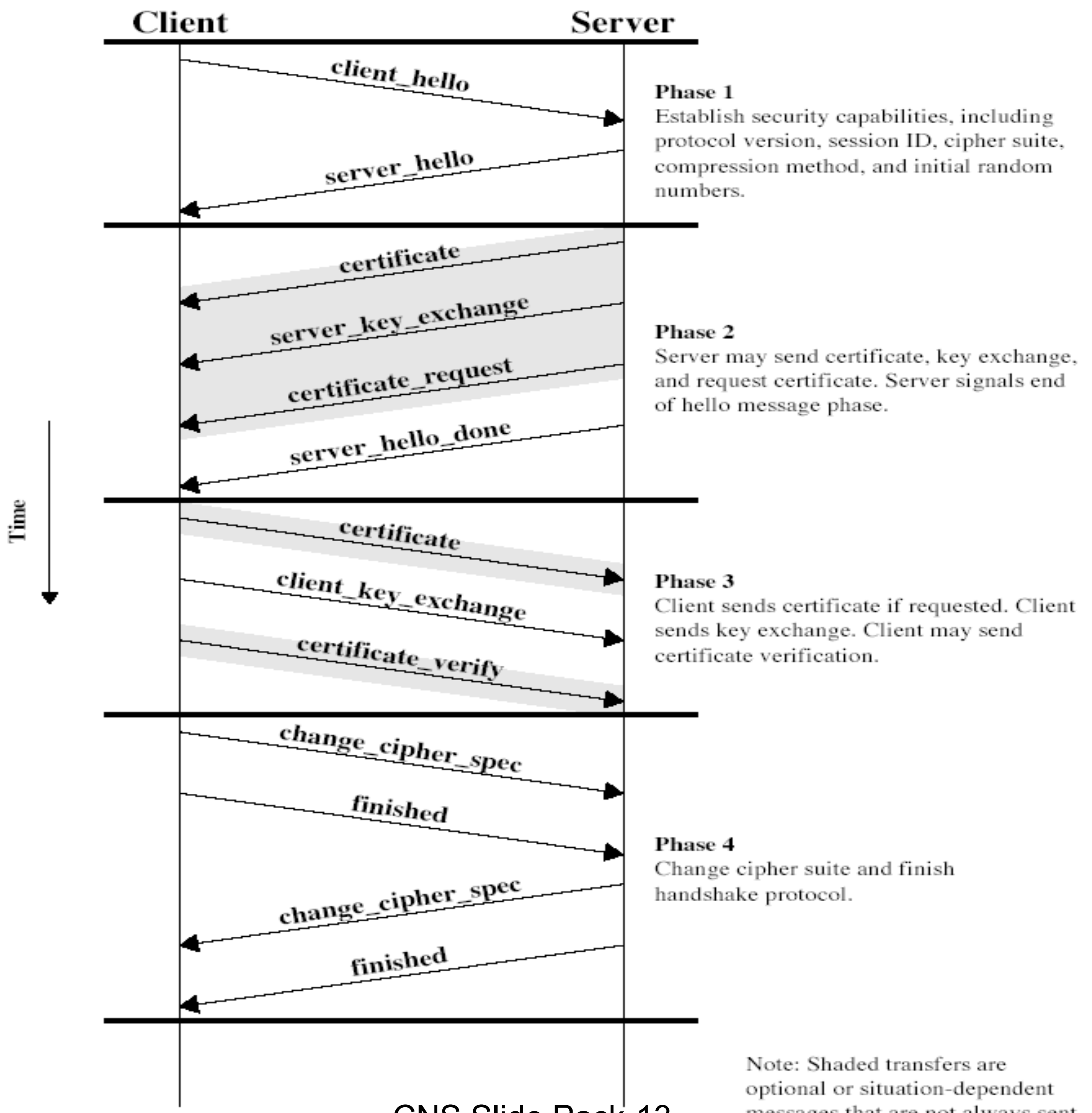
- conveys SSL-related alerts to peer entity
- severity
  - two possibilities: *warning* or *fatal* (close connection)
- specific alert
  - fatal: unexpected message, bad record mac, decompression failure, handshake failure, illegal parameter
  - warning: close notify, no certificate, bad certificate, unsupported certificate, certificate revoked, certificate expired, certificate unknown
- compressed & encrypted like all SSL data

SSL Handshake Protocol

Most complex part of SSL

- 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

SSL Handshake Protocol



Client
Server
client_hello
server_hello
certificate
server_key_exchange
certificate_request
server_hello_done
certificate
client_key_exchange
certificate_verify
change_cipher_spec
finished
change_cipher_spec
finished
Time
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.
Phase 4
Change cipher suite and finish handshake protocol.
Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

Handshake protocol

4 steps

1. Hello: determine security capabilities
2. Server sends certificate, asks for certificate and starts exchange session keys
3. Client sends certificate and continues exchanges of keys
4. End of handshake protocol: encoded methods changes

Note: some requests are optional

clear separation between handshake and the rest (to avoid attacks)

Handshake: parameters

message type (1st byte of payload)	parameters
Hello-request	null [may be sent by server at any time: notification that client should begin negotiation process anew by sending a client hello message when convenient; this message is ignored by client in some cases]
Client-hello	version, 32-bit timestamp + 28 random bytes (nonce), sessionID, cipher suite and compression method
Server_hello	<same as Client_hello>
Certificate	X.509v3 chain of certificates
Server_key_exchange	info, signature of mess.
Certificate_request	type of cert., authority
Server_done	null
Certificate_verify	signature of certificate
Client_key_exchange	info, signature of mess.
Finished	hash of all exchanged messages (integrity of handshake protocol)

Handshake Protocol - step 1

Initialization

➔ : Client_hello: client to server

- Version = highest SSL version used by client
- 32-bit timestamp + 28 bytes random (a pseudo number generator is required)
- sessionID: = 0 new connection in new session; ≠0 update previous connection
- Cipher suite: list that contains the combinations of cryptographic algorithms supported by the client, in decreasing order of preference. Each element of the list (each cipher suite) defines both a key exchange algorithm and a CipherSpec.
- Compression algorithms: ordered sequence of acceptable algorithms

🡐 : Server_hello: server to client

- same as all above (if sessionID of client = 0 generates new sessionID)

Cipher suite

Algorithms for key exchange
- RSA : session key is encoded with server public key
- Diffie-Hellman (several versions)
  - Fixed
  - Ephemeral
  - Anonymous
- Fortezza

CipherSpec
- Crypto algorithm (either a stream algo or a block algo)
- MAC algorithm
- Hash (in byte): 0, 16 (for MD5), 20 (for SHA-1)
- Key material – info used to generate session keys
- Info for initializing CBC (initial vector)

Fixed Diffie-Hellman

- Diffie-Hellman key exchange in which server's certificate contains Diffie-Hellman public parameters signed by the certificate authority (CA)
- Client provides its Diffie-Hellman public key parameters either in a certificate, if client authentication is required, or in a key exchange message
- This method results in a fixed secret key between two peers, based on the Diffie-Hellman calculation using the fixed public keys

Ephemeral Diffie-Hellman

- Used to create ephemeral (temporary, one-time) secret keys. In this case, the Diffie-Hellman public keys are exchanged, signed using the sender's private RSA or DSS key.
- The receiver can use the corresponding public key to verify the signature. Certificates are used to authenticate the public keys.
- This would appear to be the most secure of the three Diffie-Hellman options because it results in a temporary, authenticated key.

Anonymous Diffie-Hellman

- The base Diffie-Hellman algorithm is used, with no authentication.
- Each side sends its public Diffie-Hellman parameters to the other, with no authentication. This approach is vulnerable to man-in-the-middle attacks, in which the attacker conducts anonymous Diffie-Hellman with both parties.

Handshake Protocol - step 2

Server authentication and key exchange

Server to client

- Certificate: X.509 certificate chain (optional)
- Server_key_exchange (optional)
  - a signature is created by taking the hash of a message and encrypting it with the sender's private key. In this case the hash is defined as

    hash(ClientHello.random || ServerHello.random || ServerParams)

  - So the hash covers also the two nonces from the initial hello messages. ServerParams will be specified in the next slides
- Certificate_request: (optional)
- Server_hello_done: I am done and I wait for answers

Certificate message

- The server begins this phase by sending its certificate, if it needs to be authenticated; the message contains one (or a chain of) X.509 certificates.
- The certificate message is required for any agreed-on key exchange method except anonymous Diffie-Hellman.
  - If fixed Diffie-Hellman is used, this certificate message functions as the server's key exchange message because it contains the server's public Diffie- Hellman parameters.

server_key_exchange not needed

A server_key_exchange message is not required in two instances:

(1) The server has sent a certificate with fixed Diffie-Hellman parameters, or

(2) RSA key exchange is to be used.

server_key_exchange needed

- Anonymous Diffie-Hellman. Message content consists of the two global D.H. values (a prime number and a primitive root of that number) plus the server's public D.H. key
- Ephemeral Diffie-Hellman. Message content includes the three D.H. parameters provided for anonymous D.H. plus a signature of those parameters.
- RSA key exchange, in which the server is using RSA but has a signature-only RSA key. Server creates a temporary RSA public/private key pair and use server_key_exchange message to send public key. Message content includes the two parameters of the temporary RSA public key (exponent and modulus) plus a signature of those parameters.

Handshake Protocol - step 3

Client authentication

- Client verifies server certificates and parameters
- Client to server

  Client Certificate and info to verify it: (if asked)

  Message for key exchange (Client_key_exchange)

Handshake Protocol - step 4

End: go to next phase, cipher_spec

Client to server

- Message: Change_cipher_spec
- Finished message under new algorithms, keys (new cipher_spec)

Server sends back

- Message: Change_cipher_spec
- Finished message under new algorithms, keys (new cipher_spec)

Change_cipher_spec

This command indicates that the contents of subsequent SSL record data sent by the client during the SSL session will be encrypted. The 5-byte SSL record headers are never encrypted.

SSL & TLS

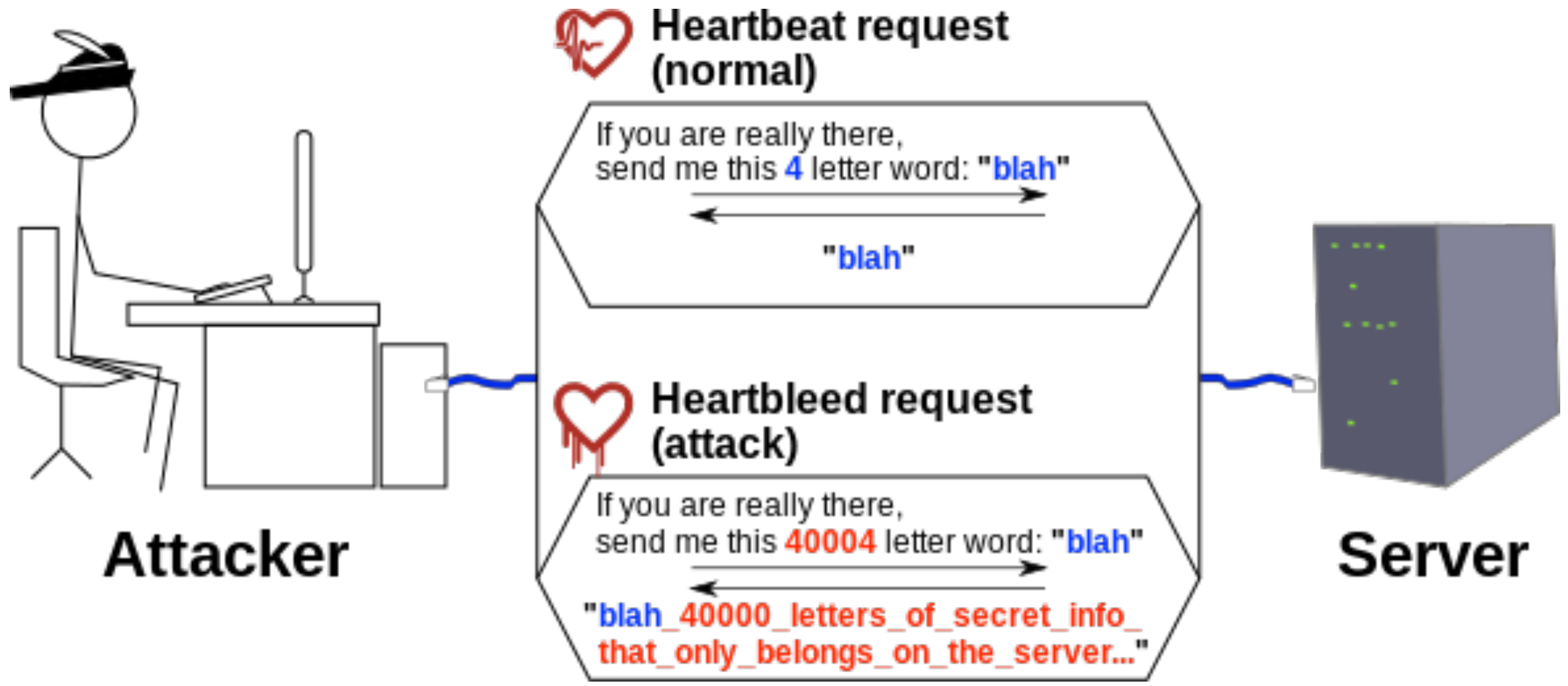
- IETF standard RFC 2246 similar to SSLv3
- with minor differences
  - in record format version number
  - uses HMAC for MAC
  - a pseudo-random function expands secrets
  - has additional alert codes
  - some changes in supported ciphers
  - changes in certificate negotiations
  - changes in use of padding

Paying in the Web: SSL

- SSL and credit card are used for paying
  - simple
  - no need of specialized software
  - compliant with credit card mechanisms
  - most used method for paying in the web
- Problems
  - malicious sellers have info on clients
  - clients can in principle refuse to pay (there is no signature)
  - many disputes (20%- 60%)
  - expensive method for the shop

SSL/TLS: heartbeat & heartbleed

- severe vulnerability in OpenSSL allowing attackers obtaining huge amounts of data from the "secure" server, by directly accessing to its memory
- according to some sources it was known since 2012
- fixed in April 2014



Heartbeat request (normal)
If you are really there, send me this 4 letter word: "blah"
"blah"
Heartbleed request (attack)
If you are really there, send me this 40004 letter word: "blah"
"blah_40000_letters_of_secret_info_that_only_belongs_on_the_server..."
Attacker
Server

POODLE attack

- SSL 3.0 vulnerable to POODLE attack, based on MITM
  - POODLE: Padding Oracle On Downgraded Legacy Encryption
- attackers need (on average) to make 256 SSL 3.0 requests to reveal one byte of encrypted messages
- the Google Security Team discovered this vulnerability in Sept. 2014