WWW Security Protocols

Giampaolo Bella

Università di Catania
http://www.dmi.unict.it/~nas

Security protocols for the WWW

- The backbone
  - HyperText Transfer Protocol Secure (HTTPS)
  - Secure Socket Layer (SSL)
  - Transport Layer Security (TLS)
- Can be fleshed up for, e.g.
  - Email: Secure/Multipurpose Internet Mail Extensions (S/MIME)
  - Payments: 3D-Secure (exposed as Verified by Visa and Mastercard SecureCode), replacing Secure Electronic Transactions (SET), now deprecated

HTTPS

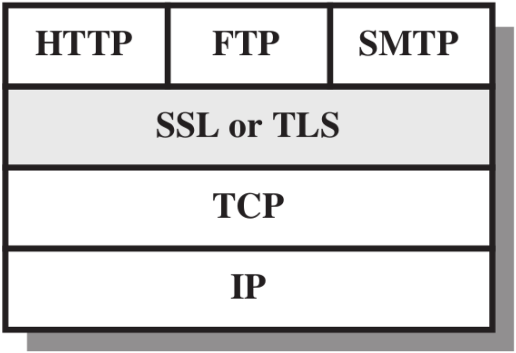
- Documented as RFC 2818 “HTTP Over TLS”
- Uses port 443 rather than 80 of HTTP
- Encrypts: contents, forms, cookies, HTTP headers (*browser type and version, O.S. used...*)
- A special header sent by server to browser is HTTP Strict Transport Security (HSTS) to thwart SSL stripping attacks

SSL stripping attacks

- Version 1: prevented by HSTS
  - User wants `https://www.securesite.com`
  - MitM downgrades response to `http://www.securesite.com`
- Version 2: not prevented by HSTS
  - User wants `https://www.securesite.com`
  - MitM downgrades response to `http://www.securesitee.com`

SSL

Provides a secure layer between TCP/IP and applications



Reserved SSL ports

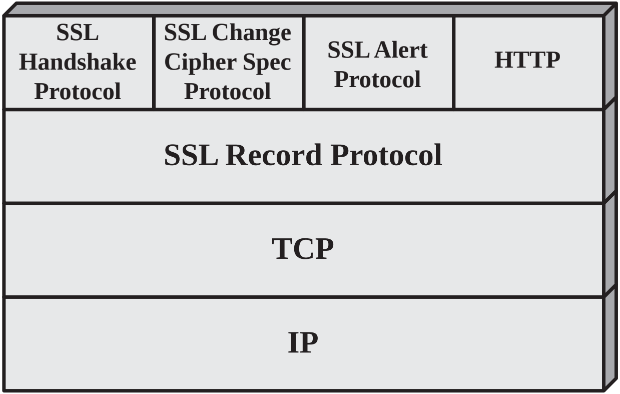
Protocol	Description	Port #
nsiiops	IIOP Name Service over SSL/TLS	261
https	HTTP over SSL/TLS	443
nntps	NNTP over SSL/TLS	563
ldaps	LDAP over SSL/TLS	636
ftps-data	FTP Data over SSL/TLS	989
ftps	FTP Control over SSL/TLS	990
telnets	Telnet over SSL/TLS	992
imaps	IMAP4 over SSL/TLS	993
ircs	IRC over SSL/TLS	994
pop3s	POP3 over SSL/TLS	995
tftps	TFTP over SSL/TLS	3713
sip-tls	SIP over SSL/TLS	5061

SSL versus TLS? Ask Microsoft!

Brief history

- 1994: Netscape deploys SSL v2 in Navigator 1.1, soon found vulnerable
- 1995: SSL 3.0 as RFC 6101 of 2011
- 1999: TLS working group of IETF standardises the protocol as TLS

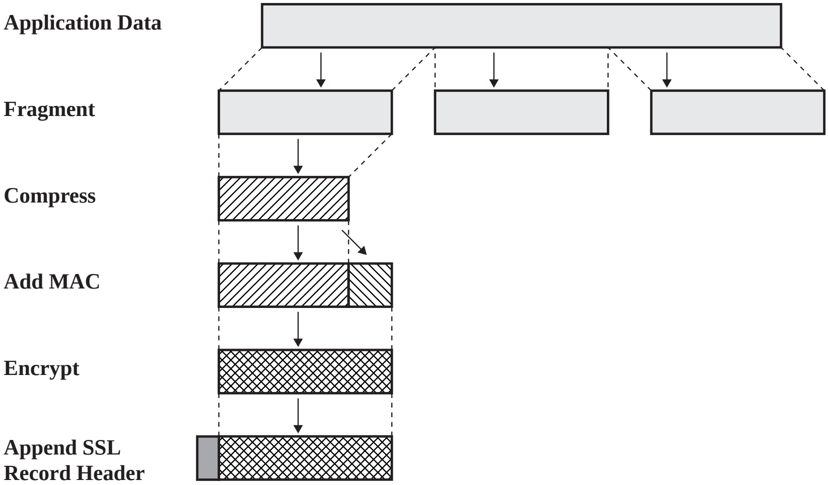
SSL protocol suite



SSL RP

SSL Record Protocol

Compact view



MAC

```
hash(MAC_write_secret ‖ pad_2‖
    hash(MAC_write_secret ‖ pad_1‖seq_num ‖
    SSLCompressed.type ‖ SSLCompressed.length ‖
    SSLCompressed.fragment))
```

where

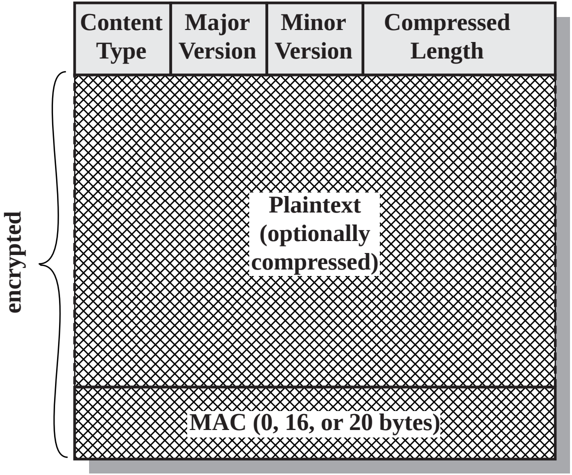
‖	=	concatenation
MAC_write_secret	=	shared secret key
hash	=	cryptographic hash algorithm; either MD5 or SHA-1
pad_1	=	the byte 0x36 (0011 0110) repeated 48 times (384 bits) for MD5 and 40 times (320 bits) for SHA-1
pad_2	=	the byte 0x5C (0101 1100) repeated 48 times for MD5 and 40 times for SHA-1
seq_num	=	the sequence number for this message
SSLCompressed.type	=	the higher-level protocol used to process this fragment
SSLCompressed.length	=	the length of the compressed fragment
SSLCompressed.fragment	=	the compressed fragment (if compression is not used, this is the plaintext fragment)

Record Header

Content type	Major version	Minor version	Compressed length

- Content type: one of the protocols of SSL
- Major version: e.g. 3
- Minor version: e.g. 0
- Compressed length: number of bytes of plaintext fragment

Record Format



SSL CCSP

Change Cipher Spec Protocol

- Trivially implements client/server snap agreement
- Consists of a message with a single byte of value 1
- Also viewed as part of SSL HP
- Causes pending state to be saved as current state
- Updates Cipher Suite field for current connection

SSL AP

Alert Protocol

- Alerts also occur over SSL RP, hence protected
- Each message consists of two bytes: level and code
- Fatal alert causes connection termination and Is Resumable set to zero; other connections continue
- Example fatal alerts: `unexpected_message`, `bad_record_mac`, `decompression_failure`
- Example warning alerts: `unsupported_cert`, `cert_revoked`, `cert_expired`

SSL HP

SSL Handshake Protocol

In short

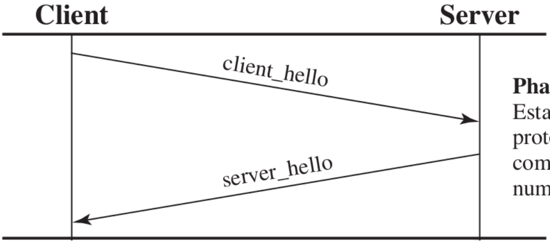
- Most complicated, actual security protocol
- Confidentiality (symmetric enc), authentication (asymmetric enc) and integrity (hashing)
- Establishes Master Secret and derives secrets from it
- Runs prior to any application data transmission
- Message format is
  - Type: one of ten message names
  - Length: message byte-length
  - Content: message fields (i.e. parameters)

1 byte	3 bytes	≥ 0 bytes
Type	**Length**	**Content**

Messages

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

Phase 1. Establish Security Capabilities



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

Client Hello parameters

Two-way agreement: client suggests, server (dis)confirms

- Client sends
  - Version: highest supported protocol version
  - Random: own fresh nonce to prevent replay
  - Session ID: zero for new session or session id to resume
  - Cipher Suite: list of (Cipher Spec, Key Exchange Algo)
  - Compression Method: list of those supported

Server Hello parameters

Two-way agreement: client suggests, server (dis)confirms

- Server sends
  - Version: lower of the versions suggested by the client and the highest supported by the server
  - Random: own fresh nonce to prevent replay
  - Session ID: if client's non zero, then server may opt to copy it, else server generates new one
  - Cipher Suite: Cipher Spec and Key Exchange Algo chosen
  - Compression Method: single method chosen

Cipher Suite components

1. Cipher Spec
    1. Cipher Algorithm: data encryption algo
    2. MAC Algorithm: hashing
    3. Cipher Type: stream or block

    4. Is Exportable: (from the US), flag
    5. Hash Size: 0 or 16 (MD5) or 20 (SHA-1) bytes
    6. Key Material: a sequence of bytes for subsequent key generation
    7. IV Size: size of the Initialization Value for CBC encryption

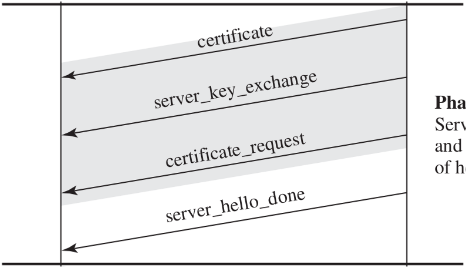
Cipher Suite components

2. Key Exchange Algorithm chosen among
   - RSA key-exchange: session key encrypted with server pk
   - Anonymous Diffie-Hellmann: traditional version, MITM
   - Ephemeral Diffie-Hellmann: public parameters authenticated by digital signature
   - Fixed Diffie-Hellmann: DH public parameters fixed, derived from (client and) server certificates
   - Fortezza: now deprecated

Possible Cipher Suites

CipherSuite	Key Exchange	Cipher	Hash
SSL_NULL_WITH_NULL_NULL	NULL	NULL	NULL
SSL_RSA_WITH_NULL_MD5	RSA	NULL	MD5
SSL_RSA_WITH_NULL_SHA	RSA	NULL	SHA
SSL_RSA_EXPORT_WITH_RC4_40_MD5	RSA_EXPORT	RC4_40	MD5
SSL_RSA_WITH_RC4_128_MD5	RSA	RC4_128	MD5
SSL_RSA_WITH_RC4_128_SHA	RSA	RC4_128	SHA
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RSA_EXPORT	RC2_CBC_40	MD5
SSL_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA_CBC	SHA
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	RSA_EXPORT	DES40_CBC	SHA
SSL_RSA_WITH_DES_CBC_SHA	RSA	DES_CBC	SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES_EDE_CBC	SHA
SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	DH_DSS_EXPORT	DES40_CBC	SHA
SSL_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES_CBC	SHA
SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES_EDE_CBC	SHA
SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	DH_RSA_EXPORT	DES40_CBC	SHA
SSL_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES_CBC	SHA
SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES_EDE_CBC	SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	DHE_DSS_EXPORT	DES40_CBC	SHA
SSL_DHE_DSS_WITH_DES_CBC_SHA	DHE_DSS	DES_CBC	SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES_EDE_CBC	SHA
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	DHE_RSA_EXPORT	DES40_CBC	SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES_CBC	SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES_EDE_CBC	SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5	DH_anon_EXPORT	RC4_40	MD5
SSL_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4_128	MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA	DH_anon	DES40_CBC	SHA
SSL_DH_anon_WITH_DES_CBC_SHA	DH_anon	DES_CBC	SHA
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES_EDE_CBC	SHA

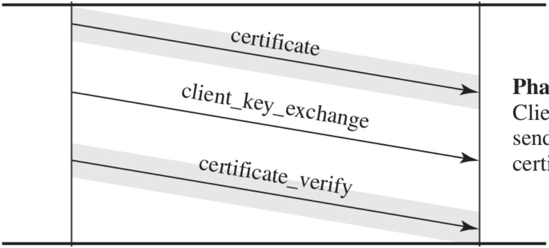
Phase 2.
Server Authentication and Key Exchange



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

- `certificate`, `certificate_request`: obvious
- `server_key_exchange`: half of Key Exchange Algo
- `server_hello_done`: tell client parameters OK
- Signatures include Random fields to thwart replays
  - DSS by SHA-1, RSA by concatenating MD5 and SHA-1

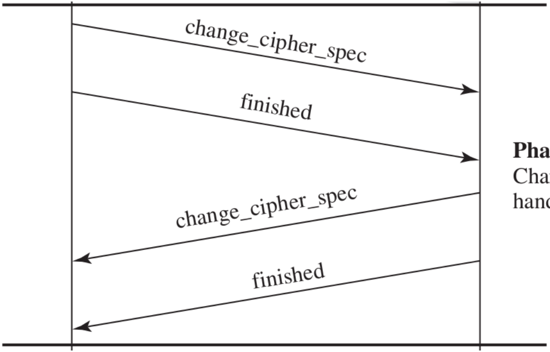
Phase 3. Client Authentication and Key Exchange



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

- `certificate`: client's certificate or alert if none
- `client_key_exchange`: 48-byte Pre-Master Secret
- `certificate_verify`: signs all traffic seen to ask explicit server's verification of client's signature
- Master Secret derived from Pre-Master Secret

Phase 4. Finish



Phase 4
Change cipher suite and finish handshake protocol.

- `change_cipher_spec`: just byte 1
- `finished`: full traffic digest using new Cipher Spec

Master Secret

$$
\begin{aligned}
\texttt{master\_secret} = \; & \texttt{MD5(pre\_master\_secret} \parallel \texttt{SHA('A'} \parallel \\
& \quad \texttt{pre\_master\_secret} \parallel \texttt{ClientHello.random} \parallel \\
& \quad \texttt{ServerHello.random))} \parallel \\
& \texttt{MD5(pre\_master\_secret} \parallel \texttt{SHA('BB'} \parallel \\
& \quad \texttt{pre\_master\_secret} \parallel \texttt{ClientHello.random} \parallel \\
& \quad \texttt{ServerHello.random))} \parallel \\
& \texttt{MD5(pre\_master\_secret} \parallel \texttt{SHA('CCC'} \parallel \\
& \quad \texttt{pre\_master\_secret} \parallel \texttt{ClientHello.random} \parallel \\
& \quad \texttt{ServerHello.random))}
\end{aligned}
$$

- 3 concatenated applications of MD5: 48 bytes
- PMS as a seed, Random values as salt

Key blocks

```
key_block = MD5(master_secret ‖ SHA('A' ‖ master_secret ‖
                ServerHello.random ‖ ClientHello.random)) ‖
            MD5(master_secret ‖ SHA('BB' ‖ master_secret ‖
                ServerHello.random ‖ ClientHello.random)) ‖
            MD5(master_secret ‖ SHA('CCC' ‖ master_secret ‖
                ServerHello.random ‖ ClientHello.random)) ‖ . . .
```

- Process continues at both ends till construction of
  - Write Key
  - Write MAC Secret

Reducing network latency

Network latency is an issue, no milliseconds to waste!

- Session: an association between a client and a server, created by SSL HP, defines crypto parameters to use over multiple connections for efficiency

- Connection: a transport according to OSI layering, peer-to-peer, transient, associated with one session

Idea

Save a session state and resume it over a new connection using pre-agreed material

Session State

1. Session ID: arbitrarily chosen by server
2. Peer Certificate: X509. 3.0 certificate of the peer
3. Compression Method: algorithm specification
4. Cipher Spec: data encryption and MAC algorithms
5. Master Secret: 48 bytes shared secret
6. Is Resumable: flag

Session resumption

- Session State saved at end of Phase 4
- Client suggests id to resume in `client_hello`
- Server has db of session id's and decides if to resume
- Phases 2 and 3 skipped; only Phases 1 and 4 run
- New key blocks calculated from Master Secret that was stored in Session State but using fresh Random values

TLS

Transport Layer Security

TLS

- Version 1.0, 1999, RFC2246
- Standardised version of SSL 3.0
- A few differences
  - Hash-based Message Authentication Code (HMAC)
  - Pseudo Random Function
  - Extra Cipher Suites
  - Extra alerts

MAC

- By standard HMAC, RFC2104

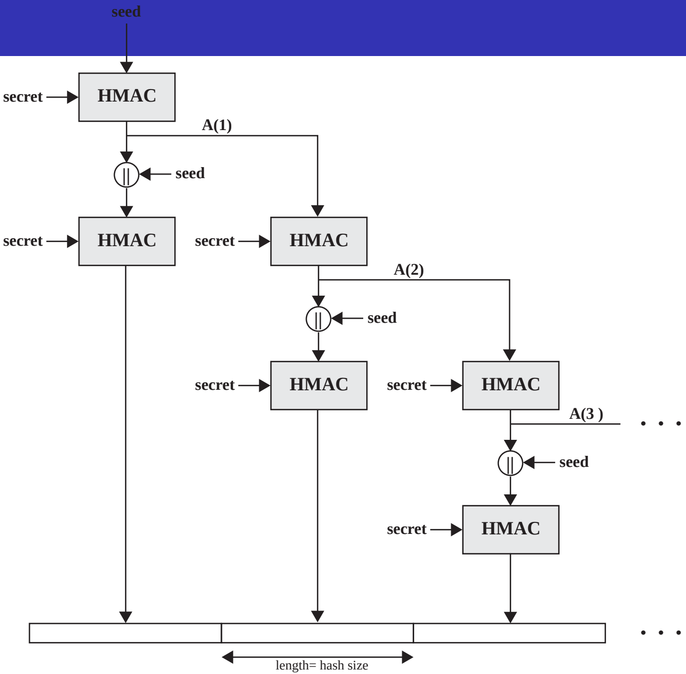
$$\mathrm{HMAC}_K(M) = \mathrm{H}[(K^+ \oplus \mathrm{opad}) \| \mathrm{H}[(K^+ \oplus \mathrm{ipad}) \| M]]$$

where

H	=	embedded hash function (for TLS, either MD5 or SHA-1)
M	=	message input to HMAC
K^+	=	secret key padded with zeros on the left so that the result is equal to the block length of the hash code (for MD5 and SHA-1, block length = 512 bits)
ipad	=	00110110 (36 in hexadecimal) repeated 64 times (512 bits)
opad	=	01011100 (5C in hexadecimal) repeated 64 times (512 bits)

- Takes M as a seed and K as a secret
- Applies a chosen hash function H

P_hash


seed
secret
HMAC
A(1)
seed
secret
HMAC
secret
HMAC
A(2)
seed
secret
HMAC
secret
HMAC
A(3)
. . .
seed
secret
HMAC
. . .
length= hash size

Pseudo Random Function

$$\text{PRF(secret,label,seed)} = \text{P\_MD5(S1,label} \parallel \text{seed)} \oplus$$
$$\text{P\_SHA-1(S2,label} \parallel \text{seed)}$$

- Parameter secret is split up into S1 and S2
- Output length controlled by reiterations in HMAC

- seed is concatenation of both Random values
- secret is PMS to oputput MS or MS for key block
- label is "master secret" to output MS or "key block" to output key block

TLS evolution

- TLS 1.0 = SSL 3.1, RFC2246, 1999
- TLS 1.1 = SSL 3.2, RFC4346, 2006
  - TLS Extensions, RFC4366, 2006
- TLS 1.2 = SSL 3.3, RFC5246, 2008

  "TLS allows extensions to follow the compression_methods field in an extensions block. The presence of extensions can be detected by determining whether there are bytes following the compression_methods at the end of the ClientHello."

TLS evolution

- TLS 1.1: No significant design changes wrt TLS 1.0
- TLS 1.2

Aug'04 MD5 found to suffer collisions

Dec'08 Sotirov-Stevens exploit MD5 to create rogue CA!

Aug'08 RFC5246 stated

- *"Substantial cleanup to the client's and server's ability to specify which hash and signature algorithms they accept."*

Security economics

Why could MD5 claim more victims even subsequently, e.g. Flame malware of 2012?

TLS evolution

- TLS 1.2: Significant design changes wrt TLS 1.0

Aug'08 RFC5246 stated

- *"Substantial cleanup to the client's and server's ability to specify which hash and signature algorithms they accept."*
- *"The MD5/SHA-1 combination in the digitally-signed element has been replaced with a single hash. Signed elements now include a field that explicitly specifies the hash algorithm used."*
- *"The MD5/SHA-1 combination in the pseudorandom function (PRF) has been replaced with cipher-suite-specified PRFs."*
- *"All Cipher Suites in this document use P_SHA256."*

Our days

Jun'15	IETF deprecates SSL 3.0 in RFC7568
Jun'16	Google stops using RC4 and SSL 3.0
Jul'16	TLS 1.3 becomes a working draft of IETF
Jan'17	SHA-1 certificates deprecated
Sep'17	Google distrusts Symantec
Jun'15	IETF approves TLS 1.3 as Internet standard

Best-known Attacks

2011 Beast chosen plaintext attack
up to TLS 1.0

2012 Crime cookie hijacking
up to early 2012 browsers

2013 Breach confidentiality attack
up to early 2013 browsers

2014 Heartbleed server memory overread
up to 2014 OpenSSL

2014 Poodle a downgrade attack to SSL 3.0
up to TLS 1.1

SSL/TLS sources

- Opplinger's "SSL and TLS: Theory and Practice", Artech House
- Sherif's "Protocols for Secure Electronic Commerce", CRC Press
- SSL/TLS and PKI History, `https://www.feistyduck.com/ssl-tls-and-pki-history/`