DRONACHARYA
College of Engineering

Computer Science & Engineering

Data Communication and Computer Networks

(MTCSE-101-A)

Web security:
SSL and TLS

What are SSL and TLS?

- SSL – Secure Socket Layer
- TLS – Transport Layer Security
- both provide a secure transport connection between applications (e.g., a web server and a browser)
- SSL was developed by Netscape
- SSL version 3.0 has been implemented in many web browsers (e.g., Netscape Navigator and MS Internet Explorer) and web servers and widely used on the Internet
- SSL v3.0 was specified in an Internet Draft (1996)
- it evolved into TLS specified in RFC 2246
- TLS can be viewed as SSL v3.1

The Idea

- Encrypt the web traffic between two sites, so no one can listen in and get credit card numbers

- Uses something called “Secure Sockets Layer” (SSL)

SSL components

- SSL Handshake Protocol
  - negotiation of security algorithms and parameters
  - key exchange
  - server authentication and optionally client authentication
- SSL Record Protocol
  - fragmentation
  - compression
  - message authentication and integrity protection
  - encryption
- SSL Alert Protocol
  - error messages (fatal alerts and warnings)
- SSL Change Cipher Spec Protocol
  - a single message that indicates the end of the SSL handshake

The Implementation

- The secure web site includes a digital certificate signed by some certificate authority. The certificate includes the server name, its public key, IP number, and an expiration date. It is typically signed with a 1024 bit key by the CA
- The list of certificate authorities that you trust to identify people is available in Netscape by clicking on the lock icon at top; in IE, Internet Options->Content

How It Works

- The browser reads the site certificate; if it is signed by one of the trusted certificate authorities, browser accepts the certificate as valid
- If the certificate is signed by some unknown certificate authority, Netscape will ask you if you want to trust the guy who signed it

How It Works (Basic Protocol)

- The browser negotiates a secure session using something like the following protocol:

1: A->B: hello
2: B->A: Hi, I'm Bob, bobs-certificate
3: A->B: prove it
4: B->A: Alice, This Is bob
{ digest[Alice, This Is Bob] } bobs-private-key
5: A->B: ok bob, here is a secret {secret} bobs-public-key
6: B->A: {some message}secret-key

How It Works

- Step 1: your browser introduces itself to the secure server
- Step 2: the server responds by sending back a message with the certificate included
- Step 3: Your browser tells the secure site to prove its identity, that it really is who it says it is.

How It Works

- Step 4: The secure server proves who it is by creating a message for the browser, generating a “fingerprint” of that message, and encrypting the “fingerprint” with the private key that is matched to the public key in the certificate. The browser generates the “fingerprint” for the message itself, then decrypts the “fingerprint” generated by the server using the public key provided in the certificate.

How It Works

- At this point the browser is sure that the server is how it says it is. It can send it secret messages encrypted with the public key provided in the certificate. The server (and only the server) can decrypt these messages, because only it has the private key.

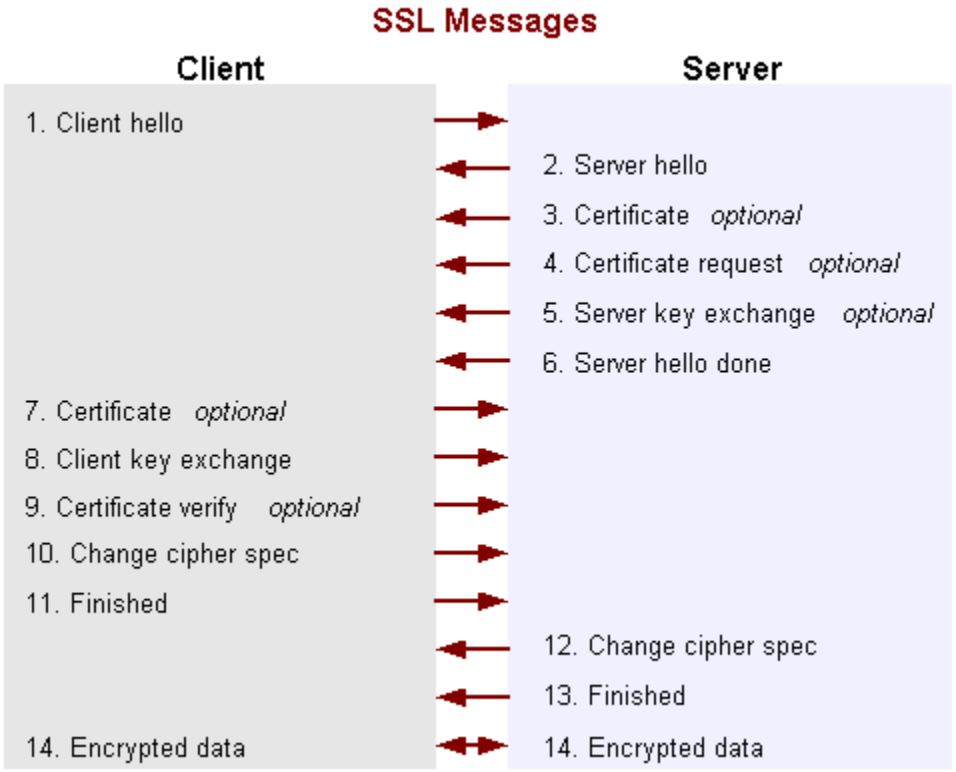
How It Works

- At this point what typically happens is that the browser generates a *session key* using a completely different encryption algorithm. A new session key is generated for every connection; this does not have to be a public key algorithm. You can use any encryption algorithm you like; usually a faster conventional, non-PK algorithm is used. This is usually 40 or 128 bits long in Netscape.

How It Works

- You'll use a completely different key for encrypting traffic to the web site every time you connect. This makes cracking communication more difficult; you need to discover the keys for every session rather than just one key.

How SSL Works: the *Handshake* in Detail



Server certificate and key exchange messages

- **certificate**
  - **required for every key exchange method**
  - **contains one or a chain of X.509 certificates**
  - **may contain**
    - **public RSA key suitable for encryption, or**
    - **public RSA or DSS key suitable for signing only, or**

- **server_key_exchange**
  - **sent only if the certificate does not contain enough information to complete the key exchange (e.g., the certificate contains an RSA signing key only)**
  - **may contain**
    - **public RSA key (exponent and modulus), or**
  - **digitally signed**
    - **if DSS: SHA-1 hash of (client_random | server_random | server_params) is signed**
    - **if RSA: MD5 hash and SHA-1 hash of (client_random | server_random | server_params) are concatenated and encrypted with the private RSA key**

Certificate request and server hello done msgs

- certificate_request
  - sent if the client needs to authenticate itself
  - specifies which type of certificate is requested (rsa_sign, dss_sign, …)
- server_hello_done
  - sent to indicate that the server is finished its part of the key exchange
  - after sending this message the server waits for client response
  - the client should verify that the server provided a valid certificate and the server parameters are acceptable

Client authentication and key exchange

- **certificate**
  - **sent only if requested by the server**
  - **may contain**
    - **public RSA or DSS key suitable for signing only, or**
- **client_key_exchange**
  - **always sent**
  - **may contain**
    - **RSA encrypted pre-master secret, or**

- **certificate_verify**
  - **sent only if the client sent a certificate**
  - **provides client authentication**
  - **contains signed hash of all the previous handshake messages**
    - **if DSS: SHA-1 hash is signed**
    - **if RSA: MD5 and SHA-1 hash is concatenated and encrypted with the private key**

    MD5(master_secret | pad_2 | MD5(handshake_messages | master_secret | pad_1))

    SHA(master_secret | pad_2 | SHA(handshake_messages | master_secret | pad_1))

Finished messages

- finished
  - sent immediately after the change_cipher_spec message
  - used to verify that the key exchange and authentication was successful
  - contains the MD5 and SHA-1 hash of all the previous handshake messages:

    MD5(master_secret | pad_2 | MD5(handshake_messages | sender | master_secret | pad_1)) |

    SHA(master_secret | pad_2 | SHA(handshake_messages | sender | master_secret | pad_1))

    where “sender” is a code that identifies that the sender is the client or the server

How SSL Achieves *Confidentiality*

- Create a secret key
  - Based on information generated by the client with a secure random number generator
- Use public keys to exchange the secret key
  - The server sends its public key to the client
  - The client encrypts the secret key with the server's public key and sends it to the server
  - The server decrypts the secret key information with the server’s private key
- Encrypt and decrypt data with the secret key
  - The client and server use the negotiated algorithm

Logistics

- To set up a secure server you need to get a certificate. Most people go to verisign (www.verisign.com). Verisign charges $350 for a certificate for one web site; it is tied to that web site name (eg www.nps.navy.mil). For commercial entities they do a search of Dun & Bradstreet to ensure who you are. This is good for one year.
- Other Certificate Authorities are www.thawte.com ($125), or any of those listed as signers in Netscape. You can set up your own CA and sign your own certificates.

Web Server Configuration

- Secure servers listen on a different port by default than normal web servers. A new instance of the program should listen on port 442 rather than port 80.
- To configure Apache: see http://www.modssl.org. The legality of the crypto package used is questionable if used for commercial purposes; the algorithms are encumbered with patent issues

Security Achieved by the Secure Sockets Layer (SSL)

- *Confidentiality*

  Encrypt data being sent between client and server, so that passive wiretappers cannot read sensitive data.

- *Integrity Protection*

  Protect against modification of messages by an active wiretapper.

- *Authentication*

  Verify that a peer is who they claim to be. Servers are usually authenticated, and clients may be authenticated if requested by servers

TCP/IP Protocol Stack With TLS/ SSL

TCP/IP Layer	Protocol
Application Layer	HTTP, IMAP, NNTP, Telnet, FTP, etc.
Secure Sockets Layer	SSL
Transport Layer	TCP
Internet Layer	IP