Announcement

• Final exam: Wed, June 9, 9:30-11:18
• Scope: materials after RSA
• (but you need to know RSA)
• Open books, open notes.
• Calculators allowed.
We have learned …

- Symmetric encryption: DES, 3DES, AES, RC4
- Public-key encryption: RSA
- Hash: SHA-1
- MAC: CBC-MAC, CMAC, HMAC
- Digital signature: RSA
- Entity authentication: challenge and response
- Key agreement: Diffie-Hellman, RSA
- Certificate
SSL-Secure Socket Layer
SSL (Secure Socket Layer)

- TCP: provides a reliable end-to-end service.
- TCP & SSL: provides a reliable & secure end-to-end service.
- HTTPS: HTTP over SSL (or TLS)
  - Typically on port 443 (regular http on port 80)
- SSL originally developed by Netscape
- subsequently became Internet standard known as TLS (Transport Layer Security)
- SSL has two layers of protocols
SSL Architecture
SSL Record Protocol Services

- SSL Record Protocol provides two services.
  - Message integrity
    - using a MAC with a shared secret key
    - similar to HMAC but with different padding
    - hash functions: MD5, SHA-1
  - Message confidentiality
    - using symmetric encryption with a shared secret key
    - Encryption algorithms: AES, IDEA, RC2-40, DES-40, DES, 3DES, RC4-40, RC4-128
SSL Record Protocol Operation

Application Data

Fragment

$\leq 2^{14}$ bytes

(Optional; default: null)

Compress

Add MAC

Encrypt

Append SSL Record Header
SSL MAC

• Similar to HMAC, using MD5 or SHA-1.

• \( \text{HMAC}_k(m) = \text{hash}(k \oplus opad \| \text{hash}(k \oplus ipad \| m)) \)

• The SSL MAC is computed as:

\[
\text{hash(MAC_write_secret} \ |\ | \text{pad}_2 \ |\ | \text{hash(MAC_write_secret} \ |\ | \text{pad}_1 \ |\ | \text{seq_num} \ |\ | \text{SSLCompression.type} \ |\ | \text{SSLCompression.length} \ |\ | \text{SSLCompression.fragmen})
\]
SSL Handshake Protocol

• Allows server & client to:
  – authenticate each other
  – to negotiate encryption & MAC algorithms and keys

• Comprises a series of messages exchanged in phases:
  1. Establish Security Capabilities (to agree on encryption, MAC, and key-exchange algorithms)
  2. Server Authentication and Key Exchange
  3. Client Authentication and Key Exchange
  4. Finish
Stallings
Figure 17.6

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.
Phase 1: Establish Security Capabilities

Client

server_hello

←⎯⎯⎯⎯⎯⎯

client_hello

Server

• client_hello: contains a client.random and a list of cipher suites in decreasing order of preference.
• server_hello: contains a server.random and a single cipher suite selected by the server.
Cipher Suite

- Each **cipher suite** indicates a key exchange algorithm, a cipher algorithm, and a MAC algorithm.
- About 30 cipher suites have been defined, each represented by a 2-octet number.
- Users can define their own cipher suites.
- Downgrade attack: the adversary removes strong cipher suites from client_hello.
Key Exchange Algorithms

- Anonymous Diffie-Hellman
- Fixed Diffie-Hellman
- Ephemeral Diffie-Hellman
- RSA
  - Server has an RSA encryption key pair
  - Server has an RSA signature key pair
- Establish a `Pre_Master_Secret` (48 bytes)
  ⇒ `Master_Secret` (48 bytes)
  ⇒ `Keys`
Anonymous Diffie-Hellman

- The `server_key_exchange` contains the server's Diffie-Hellman public key and parameters, \((p, \alpha, \alpha^s)\).
- The client provides its public key \(\alpha^c\) in the `client_key_exchange`.
- A 48-byte `pre_master_secret` is generated from \(\alpha^{cs}\).
Fixed Diffie-Hellman

- The **certificate** contains the server's Diffie-Hellman public key and parameters \((p, \alpha, \alpha^s)\).
- The client provides its public key \(\alpha^c\) in the `client_key_exchange`.
- A 48-byte `pre_master_secret` is generated from \(\alpha^{cs}\).
Ephemeral Diffie-Hellman

- The **certificate** contains the server's signature info.
- The **server_key_exchange** contains the server's one-time \((p, \alpha, \alpha^s)\), hashed and signed.
- The client provides its public key \(\alpha^c\) in the **client_key_exchange**.
RSA Key Exchange with an encryption key

Client                                           Server

[Certificate exchange diagram]

- The certificate message contains the server's encryption key info.
- The client_key_exchange message contains a 48-byte pre_master_secret encrypted with RSA\textsubscript{$(n,e)$}. 
RSA Key Exchange with a signature key

Client                                               Server

\[\text{certificate} (n, e)\]

\[\text{server_key_exchange} (n', e')\]

\[\text{server_hello_done}\]

\[\text{client_key_exchange}\]

- The \textit{certificate} contains the server's RSA-signature info.
- The server generates a temporary RSA encryption key pair, and sends the public key info (hashed and signed) to the client in the server_key_exchange.
Client Authentication

- The server may request a certificate from the client.

- The client will send a certificate message or a no_certificate alert.
Computing the Master Secret (48 bytes)

Both sides compute the master_secret as follows:

\[
\text{master\_secret} = \\
\text{MD5} \left( \text{pms} \ || \ \text{SHA}('A' \ || \ \text{pms} \ || \ \text{client\_r} \ || \ \text{server\_r}) \right) \ || \\
\text{MD5} \left( \text{pms} \ || \ \text{SHA}('BB' \ || \ \text{pms} \ || \ \text{client\_r} \ || \ \text{server\_r}) \right) \ || \\
\text{MD5} \left( \text{pms} \ || \ \text{SHA}('CCC' \ || \ \text{pms} \ || \ \text{client\_r} \ || \ \text{server\_r}) \right)
\]

where \( \text{pms} = \text{pre\_master\_secret} \); and \( \text{client\_r} \) and \( \text{server\_r} \) are the random numbers exchanged in Phase 1.
An SSL session needs six keys:

- Client write MAC secret
- Server write MAC secret
- Client write key (for encryption)
- Server write key
- Client write IV (for CBC)
- Server write IV

These parameters are generated from the master_secret in that order as in the next slide.
Continue the following process until enough bits have been generated:

\[
\text{MD5}(\text{ms} \ || \ \text{SHA}('A' \ || \ \text{ms} \ || \ \text{client.r} \ || \ \text{server.r})) \ || \\
\text{MD5}(\text{ms} \ || \ \text{SHA}('BB' \ || \ \text{ms} \ || \ \text{client.r} \ || \ \text{server.r})) \ || \\
\text{MD5}(\text{ms} \ || \ \text{SHA}('CCC' \ || \ \text{ms} \ || \ \text{client.r} \ || \ \text{server.r})) \ || \\
\ldots
\]

where \( \text{ms} = \) master_secret; and \( \text{client.r} \) and \( \text{server.r} \) are the random numbers exchanged in Phase 1.
Phase 4: Finish

- The `change_cipher_spec` message indicates that the sender is putting the negotiated Cipher Spec into use.
- The `finished` message contains \( MD5(\cdots) \) or \( SHA(master\_secret \| pad2 \| SHA(handshake\_messages \| Sender \| master\_secret \| pad1)) \).
SSL Session and Connection

• SSL was designed to work with HTTP 1.0 which tended to open a lot of TCP connections between the same client and server.

• SSL assumes a session is a relatively long-lived thing from which many (transient) connections can be cheaply derived.

• 1 session = 1 or more connections
SSL Session

• Created by the Handshake Protocol.
• Parameters:
  – Session ID
  – Compression method
  – Cipher spec (encryption and hash algorithm)
  – Master secret
  – Is resumable
SSL Connection

- Based on an underlying TCP connection
- Associated with a session
- Parameters:
  - Server and client random
  - Server and client write MAC secret
  - Server and client write secret
  - Server and client IV (if CBC is used)
  - Sequence number
- Closed by a close_notify from each side
Session Resumption

• In handshaking, the client_hello contains a session ID:
  – If session ID = 0, start a new session
  – If session ID = x (and session x is resumable):
    • start a new connection of session x
    • (or change parameters of a current connection of session x)
    • Skip key exchange
    • Use the session’s master secret to generate keys
Master secret vs. pre-master secret

Pre_Master_Secret (48 bytes)
⇒ Master_Secret (48 bytes) (session)
⇒ Keys (connection)

• Why?
• Note: even if we use pre_master_secret to generate keys, each connection will have its own set of keys.
• Pre_master_secret
  – computed for each session during handshaking
  – could be the same for all sessions
  – is not stored (so safe)

• Master_secret
  – is stored for the lifetime of the session
  – may be compromised
  – If compromised, damage limited to a session

• What if pre_master_secret is used to generate keys?