SSL/TLS
SSL and TLS

• Secure Socket Layer (SSL)
  • Early protocol for securing web connections
  • Developed in the 90s by team led by Taher Elgamal at Netscape

• Transport Layer Security (TLS)
  • Evolution of SSL
  • Standardized by IETF
  • TLS 1.0 RFC 2246 (1999)
  • TLS 1.2 RFC 5246 (2008)

Patent issued in 1997

• ... method of encrypting and decrypting information transferred over a network between a client ... and a server ...

Source: Alexander Klink via Wikipedia
The IETF is currently considering adopting SSL as a transport protocol with security features. Netscape encourages the royalty-free adoption and use of the SSL protocol upon the following terms and conditions:

[ ... ]

you may have a royalty free license to build implementations covered by the SSL Patent Claims or the IETF TLS specification provided that you do not to assert any patent rights against Netscape or other companies for the implementation of SSL or the IETF TLS recommendation.
Goals of SSL/TLS

• End-to-End Confidentiality
  • Encrypt communication between client and server applications

• End-to-End Integrity
  • Detect corruption of communication between client and server applications

• Required server authentication
  • Identity of server always proved to client

• Optional client authentication
  • Identity of client optionally proved to client

• Modular deployment
  • Intermediate layer between application and transport layers
  • Handles encryption, integrity, and authentication on behalf of client and server applications
Certificates

- **Public key certificate**
  - Assurance by a third party that a public key is associated with an identity
  - E.g., Internet2 certifies that the public key below is associated with Brown's web server
    - Certificate fields
      - Issuer aka certificate authority
      - (Internet2)
      - Subject (Brown University; www.brown.edu)
      - Subject's public key parameters (RSA2048)
      - Subject's public key
      - Validity period (2/18/15-2/18/18)
      - Signature parameters (SHA256; RSA2048)
      - Signature

| B4 B5 BA DF 3A D5 29 FA C6 C9 16 85 EF 3D 13 F0 FA 3B A0 EE CA DA 5E 42 CB 1D 1B 82 F0 78 AA D4 09 3D EE 3D 0C 68 FA C6 42 6F 49 0C DB 9B CB 0F DB 2A D0 31 99 ED 56 5B 2E 19 7E 57 E2 C1 B0 C1 C8 4B 79 EF E7 B0 C5 90 0D AD 15 97 0D 47 56 99 A9 59 A5 06 B3 26 FB E4 F4 A8 75 04 7A 47 81 E4 DA 51 60 D7 08 C7 A8 9C F8 98 F2 B0 FD 22 5E AD 74 1D E0 AC 24 6D DB 2C 2F FB 39 76 59 A4 45 6F DF 91 88 F5 05 87 5D 4C 53 75 B2 42 D9 F4 0D C0 74 4B EE AD 63 8A 95 15 4C 6B 3F 97 5E 77 13 B9 74 C9 B2 21 FE 4A 4D 45 7F C0 38 00 A7 23 07 43 F7 6D 12 A4 89 E1 4B 85 3F 35 9C 5A 86 2B A8 16 EA AE C1 5D F2 C2 10 5D 4B 4A 2D 34 06 5A 2E 6E 7F 80 78 CD 23 3B B8 17 EF 44 65 3F CC 97 5F AA F7 E2 E9 CC F9 37 C2 DF E5 DB 47 17 D2 6F 0C 98 05 9E C0 0F 57 58 A1 AE 32 BB 70 AD BA F9 |
Chain of Trust and Revocation

- Transitive trust
  - Trust of (public key of) issuer implies trust of (public key of) subject
  - Issuer can be subject in another certificate
  - Chain of certificates
  - Root of trust?
  - Root certificates preconfigured in operating system and browser

- Certificate revocation
  - Mechanism to invalidate a previously issued certificate
  - E.g., when private key of the subject is compromised

- Revocation methods
  - List of revoked certificates posted on CA’s website
  - Online verification service provided by CA
Rogue Certificates?

- In 2011, DigiNotar, a Dutch root certificate authority, was compromised
- The attacker created rogue certificates for popular domains like google.com and yahoo.com
- DigiNotar was quickly blacklisted by browsers and filed for bankruptcy
- See the incident investigation report by Fox-IT
- Google has recently questioned the certificate issuance policies and practices of Symantec in the past several years
- Google’s Chrome browser will start distrusting Symantec’s certificates unless certain remediation steps are taken
- See post by Ryan Sleevi (Google’s Chromium team) and Symantec’s response
# TLS Building Blocks

<table>
<thead>
<tr>
<th></th>
<th>Confidentiality</th>
<th>Integrity</th>
<th>Authentication</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Setup</strong></td>
<td>Public-key encryption (e.g., RSA)</td>
<td>Public-key digital signature</td>
<td>Public-key digital signature</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(e.g., RSA)</td>
<td>(e.g., RSA)</td>
</tr>
<tr>
<td><strong>Data transmission</strong></td>
<td>Symmetric encryption (e.g., AES)</td>
<td>Cryptographic hashing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(e.g., SHA256)</td>
<td></td>
</tr>
</tbody>
</table>
• Browser sends supported crypto algorithms
• Server picks strongest algorithms it supports
• Server sends certificate (chain)
• Client verifies certificate (chain)
• Client and server agree on secret value R by exchanging messages
• Secret value R is used to derive keys for symmetric encryption and hash-based authentication of subsequent data transfer
Basic Key Exchange

- Called **RSA key exchange** for historical reasons
- Client generates random secret value $R$
- Client encrypts $R$ with public key, $PK$, of server $C = E_{PK}(R)$
- Client sends $C$ to server
- Server decrypts $C$ with private key, $SK$, of server $R = D_{SK}(C)$
Forward Secrecy

- Compromise of public-key encryption private keys does not break confidentiality of past messages
  - Attacker eavesdrops and stores communication
  - If server’s private key is compromised, attacker finds secret value $R$ in key exchange and derives encryption keys
- TLS with basic key exchange does not provide forward secrecy

\[
\begin{align*}
R &= \text{random()} \\
C &= E_{PK}(R) \\
R &= D_{SK}(C)
\end{align*}
\]
Diffie Hellman Key Exchange

Achieves forward secrecy

- Public parameters: prime p and generator g of $\mathbb{Z}_p$
- Client generates random $x$ and computes $X = g^x \mod p$
- Server generates random $y$ and computes $Y = g^y \mod p$
- Client sends $X$ to server
- Server sends $Y$ to client
- Client and server compute $K = g^{xy} \mod p$
Attacker in the Middle

Solution
- Browser and server send signed X and Y respectively
- Requires each to know the public key of the other
Attacks on DH Key Exchange

- Break the crypto
  - Discrete log is hard
  - Can be solved for short, fixed modulus and generator
- TLS uses handful of standard moduli with
  - 512 bits (legacy "export grade")
  - 1024 bits (recommended)
- TLS standard generator
  - 2 or 5
- Time and effort to crack a DH key
  - 512 bits: 30 seconds after one-week precomputation at cost of $Ks
  - 1024 bits: minutes after one-year precomputation at cost of $100Ms
- Logjam attack (Adrian+ 2015)
  - Attacker in the middle
  - Protocol flaw allowed downgrade of any DH modulus to 512 bits
  - Fixed when attack disclosed
Message Authentication Code (MAC)

- Cryptographic hash function $h(K, M)$ with two inputs:
  - Secret key $K$
  - Message $M$
- Message integrity with a MAC
  - Secret key $K$ shared by sender and recipient
  - Sender computes MAC $c = h(K, M)$ and transmits it along with message $M$
  - Receiver recomputes MAC from received message and compares it with received MAC

- Properties
  - Attacker cannot compute correct MAC for a forged message
  - More efficient than signing each message
  - Secret key can be sent in a separate encrypted and signed message

Compute $c = h(K, M)$ for sent message $M$

Accept if $d = c'$ for received message $M'$
HMAC

• Building a MAC from a cryptographic hash function is not immediate
• Because of the iterative construction of standard hash functions, the following MAC constructions are insecure:
  • $h(K \parallel M)$
  • $h(M \parallel K)$
  • $h(K \parallel M \parallel K)$

• HMAC provides a secure construction:
  $$h(K \oplus A \parallel h(K \oplus B \parallel M))$$
  • A and B are constants
  • Internet standard used, e.g., in IPSEC
  • HMAC security is the same as that of the underlying cryptographic hash function
Securing Data Transfer

• Sign and encrypt
  • The encrypted pair (message, signature) is transmitted
• MAC and encrypt
  • The encrypted pair (message, MAC) is transmitted
  • Secret key for MAC can be sent in separate message
  • More efficient than sign and encrypt
  • MAC is shorter and faster to compute than signature and verification
References

• **Logjam** attack (2015)